Schedule Database Project

CS342 Fall 2015

Bethany Armitage & Jasjot Sumal

Table of Contents

phase 4 - Oracle PL/SQL

phase 5 - Graphical User Interface Design and Implementation

Phase 1 Information Gathering and Conceptual Database Design

Section 1.1 Fact Finding Techniques and Information Gathering

This section will describe the local company this database is being designed for, and which areas of the company that will be the focus of the database.. It will detail how the research was conducted for the design, and factors that helped develop the resulting conceptual design. The resulting entities and relationships are itemized with their names, attributes, and a short description. Finally, groups of users are described with their ability to create, read, update and delete data from the database.

1.1.1 Introduction to Business

The Sequoia Sandwich Company is a local delicatessen that serves sandwiches, salads, soups, and bakery desserts. They also offer catering for groups in the form of sandwich platters, boxed lunches, along with options to add beverages, chips and side salads. There are currently four locations, three in Bakersfield: downtown, southwest, and Rosedale, plus one in the Fresno/Clovis area.

1.1.2 Description of Fact Finding Techniques

Fact finding techniques are crucial to the early stages of a database development project. They enable developers to learn about the industry they are dealing with, allowing them to identify problems that a database will help address, the business constraints the database must reflect within its own structure, and the user's priorities while using the system. These facts are necessary to acquire before modeling the database, so there is a clear understanding of the database's purpose and the information it will need to fulfill its purpose. (Connolly, 2005)

There are several fact finding techniques that are used to gather information to design a database. One technique to gather information is the preparation and distribution of surveys to employees of the company. Another very useful technique is to conduct interviews with employees to understand their operating procedure, and especially their needs for completing their task more efficiently. Unfortunately, this is also the most time-consuming technique, and therefore must be used selectively. On site shadowing of daily operations with employees is a another technique that often occurs hand-in-hand with interviewing, allowing developers to experience company needs first hand. Generally, a business's documentation is also useful to keep on hand for designing constraints. This can also aid in understanding the business current structure, and thus how the need for the database arose. Further information may be gathered about the business type in general from internet research. In some cases, prior first hand work experience or association with the company, if obtainable, is very insightful. (Connolly, 2005)

1.1.3 Focus of Database

The focus of the database is management of employee work availability and weekly scheduling. The company currently stores its employees' availability on pen and paper. The manager then references this paper to type the weekly schedule in a Microsoft Excel spreadsheet, which uses formulas for counting the current number of days that any given employee has on the schedule being created. This formula is not very comprehensive, as it will not even compensate for misspellings. The manager must manually cross check several scheduling criteria on their own. The employee must availability for the shifts and days that they are scheduled for, they must only be scheduled once per day, they cannot work days requested off, they may not work well with another employee, and can only work shifts corresponding to positions they know. The schedule created is printed out upon completion, and posted at the work location. This means employees must copy this information down for their own reference, and if the schedule is changed, employees are not aware of the changes and may either show up to a shift they were not scheduled for or not show up for a shift that they need to work.

Managing everyone's shifts is a laborious and error prone task to do by hand. Our database seeks to solve this problem by providing automation of these checks in an employee management system that will track employees and their work availability given the above constraints, so that managers can quickly and accurately schedule employees for the weekly schedule. Employees will also be able to sign into the front end with a different set of permissions, allowing them to view the weekly schedule without have the ability to modify it. This will provide them access to the latest copy of the schedule anywhere they can access the internet.

The main entities needed to represent this employee scheduling database are Employees, Users, Positions, and weekly Shifts. Although other entities will be included to represent the remainder of the shop, these are the entities needed to the database's primary purpose. The front end will focus on coordinating employee scheduling with shift openings, in addition to displaying Supply Deliveries to help employees prepare for the day.

Employee entities store basic employee information such as first name, last name, birthday, and phone number. They also store information pertinent to scheduling: dates when employees were employed and unemployed, and the maximum number of hours employees desire per week.

Position entities simply store the different positions an employee can work; the only attribute they store are the names of different positions within the company. The positions are located in the front of the house, kitchen, or neither (both). Most of the relevant information regarding positions is in their relationship with Employee entities.

Finally, shift entities store information pertaining to a particular time slot for an entire week. If the company needs to have an employee that is a cashier work from 8:00AM to 3:00PM every day of the week, they would define this and it would be stored in the shift entity. So, the company will be able to decide which shifts are necessary to fulfill for the total daily workload. The shift may define different weekend hours, because the workload is different for week

days and weekend days. If a shift is only needed during the week days and not on the weekend, the weekend hours will not be defined, and the shift will function as such.

1.1.4 Itemized Descriptions of Entity Sets and Relationship Sets

Entity Sets

The following is a general summary of entities within the database including their names, meaning, attribute names and properties. More detailed descriptions can be found in the next section.

The **availability** entity stores data about hours that current employees may be scheduled for on specific weekdays.
`availability (day, startt, endt)`

The **employee** entity stores data about current employees.
`employee (id, first_name, last_name, is_manager, birth_date, max_hours_per_week, phone_number, clock_number)`

The **employment history** entity stores employment history of current and previous employees.
`employment_history (date_employed, date_unemployed)`

The **holiday** entity stores information about dates the store is closed for a full or half day.
`holiday (date, name, type)`

The **ingredient** entity stores data regarding ingredients used to create food items
`ingredient (id, name)`

The **menu item** entity stores data regarding food items that may be sold.
`menu_item (id, name, type, price, photo)`

The **position** entity stores data about various positions that employees occupy.
`position (id, title)`

The **shift** entity defines the expected tasks that need to be scheduled.
`shift (id)`

The **supplier** entity describes which companies deliver ingredients to the company.
`supplier (id, title, delivery_days)`

The **transaction** entity stores sales transactions that have been made by the company.
`transaction (id, date)`

The **user** entity stores employee login information. This is a subclass of the employee entity.
`user (email, password, api_token)`

The **weekday_hours** entity stores the start and end times for shifts that need to be scheduled for weekdays. This is a subclass of the shift entity.
`weekday_hours (start, end)`

The **weekend_hours** entity stores the start and end times for shifts that need to be scheduled for weekends. This is a subclass of the shift entity.
`weekend_hours (start, end)`

Relationship Sets

The following is a general summary of relationships within the database including their names, meaning, attributes, cardinalities, and participation constraints.

`cannot_work_with (employee1_id, employee2_id)`
Relationship of employee to employee
N…N, partial participation

`delivered_by (ingredient_id, date_delivered, quantity)`
Relationship of ingredient to supplier
1…N, partial participation

`has_availability (employee_id, availability_id)`
Relationship of employee to availability
1…1, total participation

`has_employment_history (employee_id)`
Relationship of employee to employment history
1…N, total participation

`has_position (employee_id, position_id, date_acquired, date_removed, is_primary, is_training)`
Relationship of employee to position
N…N, total participation

`has_shifts (position_id, shift_id, task_name)`
Relationship of position to shift
1…N, total participation

`is_scheduled_for (employee_id, shift_id, date)`
Relationship of employee to shift
1…N, partial participation

`orders (employee_id, ingredient_id, date, quantity)`
Relationship of employee to ingredient
1…N, partial participation

`requests_vacation (requested_by, responed_by, is_approved, start_date, start_time, end_date, end_time)`
Relationship of employee to employee
1…N, partial participation

`sold_in (menu_item_id, transaction_id, quantity)`
Relationship of menu item to transaction
N…N, total participation

`used_in (ingredient_id, menu_item_id, quantity)`
Relationship of ingredient to menu item
1…N, total participation

1.1.5 User Groups, Data Views, and Operations

The User Groups are primarily defined by the is_manager attribute within the Employee entity. This attribute dictates access to views that allow the user to add information to the database, as opposed to only visualize it. More specifically, User Groups are defined by Employee Positions; in descending order the privilege levels are manager or employee. A manager will have more access to viewing and editing data than an employee.

Managers
- read, **employment history**
- read, create, update and delete **positions**, and which employees **have which positions**
- read, create, update and delete **shifts**
- read, create, update and delete scheduled weekly **deliveries**
- read, create, update and delete **employees**
- read, create, update and delete **employee availability**
- read, create, update and delete when employees are **scheduled**
- read, create, update and delete which employees **cannot work with** together
- read, create **transactions**
- read, create, update and delete **locations**
- read, create, update **holidays**
- read, create, update and delete available **menu items**
- read, create, update and delete available **ingredients**
- read, create, update and delete placed **orders**
- read(view requests off), create(submit requests off) and update (approve/deny) **requests off**

Employees
- read **deliveries**
- read, create **transactions**
- read their own **availability**
- read, create and update **requests off** (only for own requests)
- read weekly **deliveries**

Section 1.2 Conceptual Database Design

This section describes in detail the entities and relationships for the conceptual database. First, the entities are described. They each have a detailed description, list of attributes and more specifics on each of their attributes. Next, the relationships are described. They each have a detailed description, along with their relationship cardinality and participation constraints of the relationship.

1.2.1 Entity Set Description

Listed below are the entities with more detailed descriptions about their purpose and attributes. Next to the name of the entity in parenthesis will describe if it is a strong or weak entity. A strong entity will have a unique primary key identifier, while a weak entity will not have it's own primary key. Asterisked attributes are either a primary or foreign key. The specific type of key will be asterisked in the details. In addition to the names of the attribute; their type and range are specified. If a default value exists for the attribute it will be listed. If an attribute is nullable, it will be listed as such, otherwise it is considered not nullable. The attribute will be described as unique if two rows cannot contain the exact same values. If an attribute is not described as unique it is considered to be not unique. The attribute will be described as single or multivalued, and simple or composite.

Availability	**(weak)**
The availability entity stores which hours a given employee may work itemized by each day of the week.	
*employee_id	int, 0-999, unique, single, simple, *foreign key
day	varchar(16), single, simple
startt	time, 0:00 - 24:00, default: 0:00, single, simple
endt	time, 0:00 - 24:00, default: 0:00, single, simple

Employee	**(strong)**
The employee entity describes employee basic information, maximum hours they are willing to work a week, and whether they have manager privileges or not, and system login credentials. All employees, including prior employees are stored. Employees have two relationships to themselves: some employees cannot work with others, and employees may request specific days off of work. They also have availability for shifts and they are scheduled for shifts. Employees have one or more position, and they have an employment history associated with them. This entity is also a superclass to the User entity, which provides employees with login credentials to the scheduling system. All employees are members of the User subclass.	
*id	int, 0-999, unique, single, simple, *primary key
clock_number	int, 0-999, unique, nullable, single, simple
first_name	varchar(255), single, simple
last_name	varchar(255), single, simple
is_manager	boolean, true or false, default: false, single, simple
birth_date	date, 01/01/1915 - 01/01/2515, single, simple
max_hours_per_week	int, 0-40, default: null, nullable, single, simple
phone_number	big int, 1000000000-9999999999, default: null, nullable, single, simple

Employment History	**(weak)**
There are cases where employees may work for the company for a time, leave, and then return to the company. For example, students sometimes work only summers. Employees who have at least one column with a start date and a null end date are current employees. This also convenient when employees are rehired because their information will be accessible upon reemployment.	
*employee_id	int, 0-999, unique, single, simple, *foreign key
date_employed	date, 01/01/1915 - 01/01/2515, default: today, single, simple
date_unemployed	date, 01/01/1915 - 01/01/2515, default: null, nullable, single, simple

holiday	(strong)
holidays are where the store is closed for a full or half day, and no employees are scheduled.	
*date	date 0-999, unique, single, simple, *primary key
name	varchar(255), unique, single, simple
type	varchar(255), unique, single, simple

Ingredient	(strong)
Ingredients are what are used to prepare food items. The company uses several different ingredients in different combinations. Sometimes ingredients are not available. Examples would be lettuce, tomato, various meats, various cheeses etc.	
*id	int, 0-999, unique, single, simple, *primary key
name	varchar(255), unique, single, simple

Menu Item	(strong)
Menu items are made of several ingredients, and one or more menu items are sold in a transaction. They are the means by which the sandwich shop earns its profit, and must therefore be watched closely. Relationships with other entities helps track the success of any given menu item, and thereby its current.	
*id	int, 0-999, unique, single, simple, *primary key
name	varchar(255), unique, single, simple
type	varchar(255), single, simple
price	number(* ,2), 0.00-99.99, single, simple
photo	lob, 0 gigabytes -2 gigabytes, default: 0, single, simple

Position	(strong)
The position entity stores which types of positions employees may assume. The company has different types of workers who have different tasks. Different positions will have different available shifts. For example, an employee may be a cashier, food prep, janitor etc. It is possible for an employee to be trained in more than one position. The position has a location such as dining room, kitchen, or none.	
*id	int, 0-999, unique, single, simple, *primary key
title	varchar(255), unique, single, simple
location	varchar(255), single, simple

Shift	(strong)
The shift entity describes recurring start and end times for when employees are supposed to show up for work. There may be different hours for week days and weekend days, hence the Weekday Hours and Weekend Hours subclasses. If a shift must be worked during the weekdays, but not weekends, it will be a member of the Weekday Hours subclass, but not Weekend Hours. The converse is true as well, and a shift can be a member of both subclasses, but must be a member of at least one. For example, if the company needs one cashier to work everyday in the morning, they would create a shift so that a cashier may be scheduled every day in the morning. They may choose for the cashier to work at 8:00 AM and leave at 3:00 PM on weekdays and 9:00 AM to 4:00 PM on weekend days. An employee may have availability for shifts and may be scheduled for shifts.	
*id	int, 0-999, unique, single, simple, *primary key

Supplier	(strong)
The supplier entity describes current recurring deliveries by suppliers. The company has several supply shipments a week from several other companies. The expected delivery days will affect the expected daily workload for employees.	
*id	int, 0-999, unique, single, simple, *primary key
title	varchar(255), unique, single, simple
delivery_days	varchar(255), default: “”, single, simple

Transaction	(strong)
A transaction consists of menu items, and is added to the database when a customer purchases items. Each transaction is stored with associated dates, allowing the potential for data visualization of sales over time. Companies will use this entity to track their success over time, as well as the success of specific items. This allows for the company to make informed decisions regarding employment during specific times of the year, shifts available for schedule during specific days of the week, and primarily which food items to offer and when they should be offered.	
*id	int, 0-99999, unique, single, simple, *primary key
date	date, 01/01/1915 - 01/01/2515, default: null, nullable, single, simple

1.2.2 Relationship Set Description

Listed below are the relationships with more detailed descriptions about their purpose and usage. Next to the name of the relationship in parenthesis will describe the entities being related. Below the description will be listed the cardinality, participation constraints, and descriptive fields of the relationship. In addition to the names of the descriptive fields; their type and range are specified. If a default value exists for the field it will be listed. If a field is nullable, it will be listed as such, otherwise it is considered not nullable. The field will be described as unique if two rows cannot contain the exact same values. If a field is not described as unique it is considered to be not unique. The field will be described as single or multivalued, and simple or composite.

Cannot Work With	**(Employee - Employee)**
Not all employees work well together; there may be disputes or personality types may clash, for example. Whatever the reasons, it works well to know which employees do not work well together to keep the work flow as smooth as possible. This relationship tracks which employees an employee should not work with, allowing the database to automatically restrict employees from being scheduled together.	
Cardinality	M… N
Participation Constraint	Partial for both employees being tracked, and those employees an employee cannot work with. Not all employees may have difficulties working with others, and an employee may not have difficulties with every other employee.

Delivered By	**(Ingredient - Supplier)**
A supplier may deliver ingredients used to make menu items, in which case the company should know which ingredients are being delivered. This allows the company to track its own consumption and the sources of food issues. This relationship simply associates deliveries with ingredients, and stores the delivery date.	
Cardinality	N … 1
Participation Constraint	Partial for ingredients and total for suppliers. Some deliveries bring in supplies other than food, and so may not be associated to ingredients. However, all ingredients must arrive from a supplier, so all ingredient entities will relate to a supplier.
Descriptive Fields:	Descriptive Field Qualities:
date_delivered	date, 01/01/1915 - 01/01/2515, default: null, nullable, single, simple
quantity	int, 0-127, default: 0, single, simple

Has Availability	**(Employee - Availability)**
This relationship determines the initial availability of employees for any given day, and is thus be the first relationship managers need to schedule their employees for shifts. Since an employee's availability may change on a day to day basis, it works best to directly determine which shifts an employee can and cannot work at any given point in time. By querying this relationship with 'Requests Off' and 'Has Position', it will be determined which Shift entities an Employee can be related to through the 'Is Scheduled For' relationship.	
Cardinality	1 … (1, N)
Participation Constraint	Total for employees and availability. Every employee must have an availability for work. If an employee is not currently active, their availability is 0:00-0:00 for all days, which means they have no available hours.

Has Employment History	**(Employee - Employment History)**
Every business must know who has worked for it, and at what point in time this employment occurred. This information may be reviewed at any time by tax collection or other government agencies, by other employers, or by insurance agencies, just to name a few groups. This relationship simply links the an employee entity to their associated employment histories. The cardinality is 1:N, since an employee may leave the company and return at a later point in time, requiring multiple employment histories for tracking their total history with the company.	
Cardinality	1 … N
Participation Constraint	Total for both employee entities and employment history entities. Every employee must relate to at least one employment history to be an employee of the company. Likewise, every employment history must associate with at least one employee to have a purpose in the database.

Has Position	(Employee - Position)
The sandwich shop has many positions an employee can fulfill, such as cashier, cold-prep, or janitor. For managers to effectively schedule employees, a system should be in place to automatically determine which employees have been trained, or are currently training for, the positions of shifts currently open. This relationship tracks what positions an employee is able to work, which in association with employee's 'Has Availability For' relationship, alters when a manager can schedule them. Additionally, the relationship helps managers make scheduling decisions by tracking whether or not a position is an employee's primary position, and whether or not that employee is training, allowing trainees to be scheduled with other experienced employees working the same position.	
Cardinality	N … N
Participation Constraint	Total for both employees and positions. Every employee must have at least one position to be employed with the company, but may also be trained for multiple positions. Every position must have at least one employee able to work that position for the company to function, and a position may have multiple employees trained to fulfill its position. If the only trained employee for a position quits, another employee must start training for that position.
Descriptive Fields:	Descriptive Field Qualities:
date_acquired	date, 01/01/1915 - 01/01/2515, default: null, nullable, single, simple
date_removed	date, 01/01/1915 - 01/01/2515, default: null, nullable, single, simple
is_primary	boolean, true or false, single, simple
is_training	boolean, true or false, single, simple

Has Shifts	(Position - Shifts)
Not all positions need to worked all times of the day. As a result, many positions can only be worked during certain shifts, so this relationship tracks a position's availability for shifts. This allows managers to restrict a position's shifts, so they do not have to worry about scheduling an employee for a shift that is not needed at a given time or on a given day.	
Cardinality	1 … N
Participation Constraint	Total for both positions and shifts. Every position must be available for at least one shift to have a purpose within the company, and can be available for multiple shifts. Every shift also must be associated with exactly one position to be useful.
Descriptive Fields:	Descriptive Field Qualities:
task_name	varchar(255), single, simple

Is Scheduled For	(Employee - Shift)
Using the above relationships, managers will be able to fill in the work schedule for a given week. Changes to that schedule must be tracked in the database. Once a employees are scheduled, they will be associated to shifts through this relationship, which will in turn alter the final shift schedule that employees and managers can view. This relationship will also prevent an employee from being double-scheduled for a time block, and in association with the 'Has Availability' relationship, will modify what days an employee can be scheduled.	
Cardinality	1 … N
Participation Constraint	Partial for both employees and shifts. Not all employees may be working and not all shifts may be filled during a given week.
Descriptive Fields:	Descriptive Field Qualities:
date	date, 01/01/1915 - 01/01/2515, single, simple

Orders	(Employee - Ingredient)
This relationship is meant to provide a method for tracking and adding new ingredients that may need to be added to menu items, as well as tracking the current ingredients being ordered. The association to employee entities allows users to know who placed the order for a particular ingredient, in case a chain of custody needs to be established.	
Cardinality	1 … N
Participation Constraint	Partial for employees entities and total for ingredient entities. Only a few employees, primarily managers, are involved in placing orders for ingredients, however all ingredients must be ordered at some point.
Descriptive Fields:	Descriptive Field Qualities:
date	date, 01/01/1915 - 01/01/2515, single, simple
quantity	int, 0-999, default: 0, single, simple

Requests Vacation	**(Employee - Employee)**
Employees may choose to request days off for vacation, sick leave, or other purposes. These situations will alter employees' availabilities, and therefore must be tracked. This prevents managers from scheduling employees on days they have been approved to take off. The relationship also gives managers a convenient means of tracking time-off requests as approved, denied, or still pending, and means for responding to those pending requests.	
Cardinality	1 … 1
Participation Constraint	Partial for both employees that request time off and managers that may approve requests. Not all employees may choose to request time off, and not all managers may be involved in approving or denying requests.
Descriptive Fields:	Descriptive Field Qualities:
start_date	date, 01/01/1915 - 01/01/2515, default: null, nullable, single, simple
start_time	time, 0:00-24:00, default: null, nullable, single, simple
end_date	date, 01/01/1915 - 01/01/2515, default: null, nullable, single, simple
end_time	time, 0:00-24:00, default: null, nullable, single, simple
is_approved	boolean, true or false, default: null, nullable, single, simple

Sold In	**(Menu Item - Transaction)**
A successful business must know exactly what it is selling in order to survive. A product may no longer be worth its upkeep or may be in high demand, both situations that require a response in how that product is handled by management. This relationship associates menu items to a transaction, tracking the quantity of a menu item sold through that single transaction in the process.	
Cardinality	N … 1
Participation Constraint	Partial for menu items and total for transactions. All transactions must relate to menu items, since it is not possible to sell nothing. Not all menu items need to relate to a transaction however, since customers may not ever purchase a given item.
Descriptive Fields:	Descriptive Field Qualities:
quantity	int, 0-999, default: 0, single, simple

Used In	(Ingredients - Menu Item)
The menu items unique to the company are made to order at the store. As a result, these menu items must be made from ingredients stored on site. Each store location must therefore know whether or not ingredients are available to make the items being ordered, which is the primary purpose of this relationship. Also tracked is how much of each ingredient is used in a menu item, allowing store locations to watch their total consumption, and adjust the ingredients being ordered according.	
Cardinality	1 … N
Participation Constraint	Total for ingredients and partial for menu items. All ingredients must be used in menu items to have a purpose. Not all menu items require ingredients, however, since non-unique items often arrive on site pre-packaged (e.g. chips, cookies).
Descriptive Fields:	Descriptive Field Qualities:
quantity	int, 0-999, default: 0, single, simple

1.2.3 Related Entity Set

Specialization/generalization relationships are used to describe entities that are the subclass/superclass of other entities. Specialization relationships are used to describe subclass entities that have been derived from an entity, while generalization relationships describe superclass entities derived from one or more entities. The purpose of specialization is to extract what are known as specific attributes, or attributes that are only necessary for a special category of tuples within an entity. These are placed in their own grouping, thus extracting the special category of entities themselves, allowing for these entities to receive special treatment more easily. Generalization, being the reverse of specialization, is designed to create a single entity by extracting shared attributes among different entities and creating an entity relation out of them. This gives related entities a method of being grouped together, allowing congregate operations to be performed on them more easily, and also minimizing redundancy.[1]

Two types of constraints must be defined to fully describe any specialization/generalization relationship. The first is the completeness constraint, which defines a the degree to which a superclass is a member of it subclasses, or in other words, the relative number of superclass entities that have an associated subclass entity. Total participation means every superclass entity must have at least one subclass entity associated with it, while partial participation

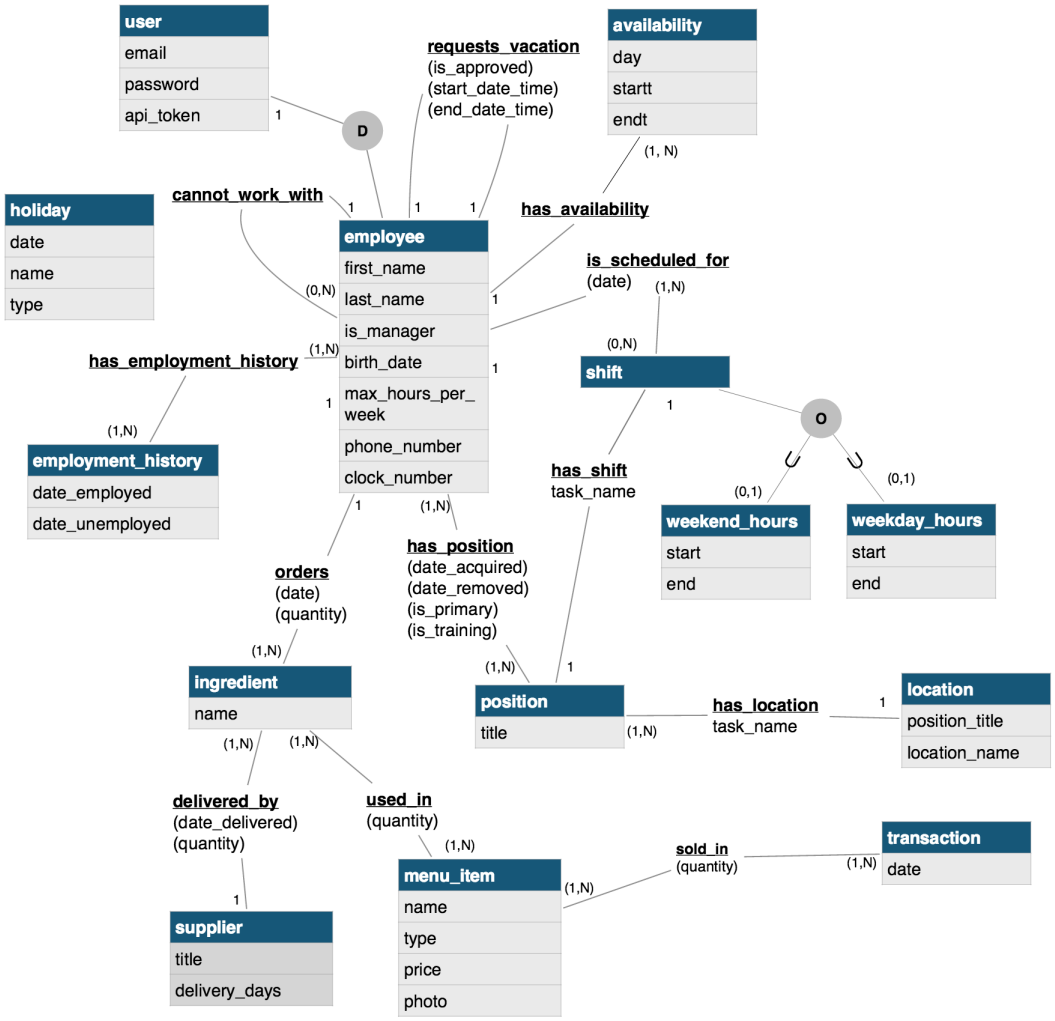
means a superclass entity can be added to the database without needing to add a corresponding subclass entity. The second constraint is the disjointness constraint, which defines the relative amount of subclasses a superclass can participate in

User	**Employee (is-A) User**
An employee is a User. They will log in to the system in order to use it. This entity is a subclass of the Employee entity. Participation Constraint: total Disjoint Constraint: overlapping	
email	varchar(255), unique, single, simple
password	varchar(255), single, simple
api_token	varchar(255), unique, single, simple

Weekday Hours	**Shift (has_A) Weekday Hours**
The start and end times for shifts that need to be scheduled for weekdays (Mondays - Fridays). This entity is a subclass of the Shift entity, allowing it to have weekday scheduling times if they are needed. Participation Constraint: total Disjoint Constraint: overlapping	
start	varchar(255), unique, single, simple
end	varchar(255), simple

Weekend Hours	**Shift (has_A) Weekend Hours**
The start and end times for shifts that need to be scheduled for weekends (Saturdays and Sundays). This entity is a subclass of the Shift entity, allowing it to have weekend scheduling times if they are needed. Participation Constraint: total Disjoint Constraint: overlapping	
start	varchar(255), unique, single, simple
end	varchar(255), simple

1.2.4 E-R Diagram



Phase 2 Convert Entity Relationship Model to Relational Model

Section 2.1 Conceptual Database (Entity Relationship) and Logical Database (Relational) Models

Two primary techniques have been developed to model databases, the Entity Relationship Model and the Relational Model. Each model is used to represent the database during a different phase of its development. This section gives a brief introduction to both models, and discusses their advantages and disadvantages.

2.1.1 Description of Entity Relationship Model and Relational Model

The Relational Model was first introduced in a 1970 paper written by Ted Codd, an IBM Engineer. This model depicts all facts in the database as relations, which are portrayed as tables containing attributes within the model. The term “relations” here represents any collection of tuples with the same attributes. The tuples themselves are analogous to facts in the database mini world. These facts may be entities or they may be relationships, but either way, facts that can be categorized together become relations in the Relational Model.

Treating both entities and relationships as relations allows the Relational Model to closely represent first-order predicate logic. This simplicity and foundation in mathematical logic attracted attention to the model, causing it to quickly gain widespread use. As a result, most database management systems (DBMS’s) developed in the 1980’s were based conceptually on the Relational model, because it allowed them to an effective and systematic method of calculating their queries. These DBMS’s, such as SQL/DS and Oracle, are now either the most heavily used DBMS’s today, or the precursors to those DBMS’s, because their foundation in mathematical logic allows for easy representation and understanding of a system’s logic database.

In response to the shortcomings of existing models, MIT researcher Peter Chen published a paper on the Entity Relationship (ER) Model in 1976. The ER Model was designed to unify the major database models in use at the time: the Network, Relational, and Entity Set models. While the Relational Model provides a high degree of data independence, meaning the organization of data is kept separate from the design of the end user application, it does little to help visualize the meaning of its relations. The Entity Set Model has shortcomings to the Relational Model, while the Network Model suffers from poor data independence but better representation of semantics. The ER Model was created to take emulate the strengths of these models, while attempting to reconcile their shortcomings, which resulted in a model popularly used today for representing a system's conceptual database. (Chen, 1976)

2.1.2 Comparison of Entity Relationship Model and Relational Model

As with the Relational Model, the ER Model represents the list of attributes for a relation, but it additionally distinguishes entities from relationships. While an entity models a specific part of a business, a relationship is used for a category of associations between entities. Both are technically represented as a table of tuples in the relational model, but they have different semantic meanings and are therefore represented distinctly in the ER Model. In addition, the ER Model can often ends up expanding the number of relationships past those that are representing in tables, because all conceptual linkages between entities are represented, regardless of their actual use in the database. For this reason, it is not practical to represent id or foreign key attributes in the ER Model, unlike the Relational Model, since it is not known which database tables will need those keys.

Additionally, it is possible to create a hierarchy of classes, termed related entity sets, in ER Model. Subclasses that can inherit attributes from other superclasses have specific notation in the ER Model that denotes some constraints specific to the inheritance, allowing viewer a better understanding of how the inheritance is meant to work. However, the Relational Model more accurately depicts how the inheritance structure exists in the database. The subclasses are modeled as weak relations which contain foreign keys to their superclasses, thus requiring that a superclass tuple exist before one can be created in a subclass. Understanding these relations as subclasses of other relations is something that must be

remembered by those implementing the database, but is not actually conceptualized in the model.

With the ER diagram, entities are represented in rectangles that may or may not link to relationships. Relationships are represented in diamonds that must form a link between two entities, which may be non-distinct. On the other hand, in the Relational Diagram, all relations are represented as tables in which the first row is the name of relation, and subsequent rows. (Chen, 1976)

Section 2.2 Conversion of Conceptual Database Model to Logical Database Model

An entity-relationship model is designed as a human readable, conceptual model, while a relational model is designed to be connected by logical connections of referential keys called foreign keys. A database is represented as a relational model in the computer. Therefore, in order to build the database in the computer, the conceptual model needs to be converted to a logical model. In this section, The conversion of entity types and relationship types to relations will be discussed, as well as techniques for keeping the integrity and consistency of what the model represents by adhering to database constraints.

2.2.1 Converting Entity Types to Relations

In order to convert an ER model to a Relational model, entity types must be converted to relations. Relations may only contain single, simple attributes, therefore, any composite or multi-valued attributes will be modified. Relations must also have a foreign or primary key included to represent how they are linked together. The primary keys are decided upon conversion of the entity types, but the inclusion of foreign keys will be decided upon conversion of relationship types.

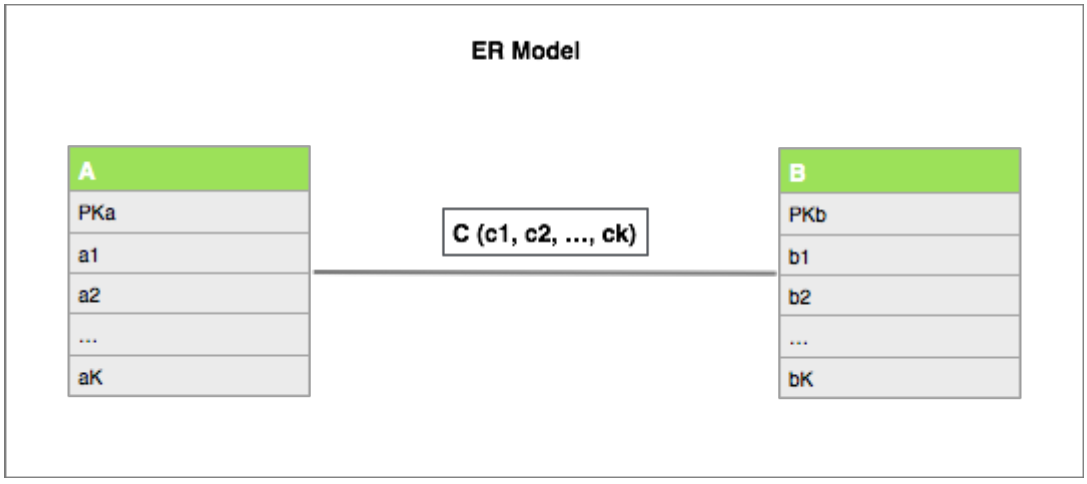
To convert a strong entity type, existing single, simple attributes are moved directly into the relation without any modification. The primary key for the resulting relation will be determined by a combination of attributes that constitute a unique identifier, or an artificial unique identifier may be created. Foreign keys may be added to strong entities upon conversion of the connecting relationships. The conversion of weak entities is similar to that of strong entities. Just as in the strong entity conversion, the single, simple attributes will be directly moved to the new relation. However, weak entities will not be given a primary key. A foreign key will be added to connect the resulting relation with what it was related to in the ER model.

Composite and multivalued attributes are mapped to relations through the same methods for both strong and weak entity types. The most common method used to include composite attributes in a relation is called “flattening” where their simple attributes are pulled out and placed in their parent entity as simple attributes. However, another option is to create a

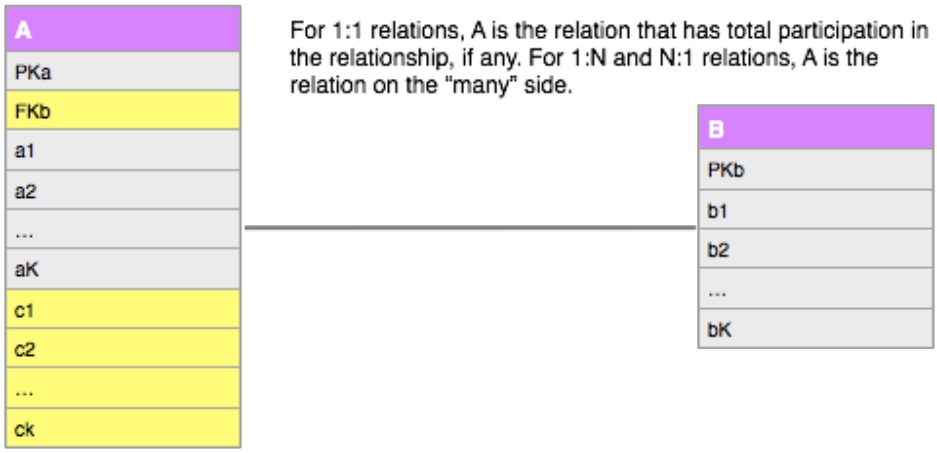
separate relation with attributes that the composite attribute contained. Making a separate entity would be a better solution for larger composite attributes, while “flattening” would be a better solution for smaller ones. The next type of attribute that must be modified are Multivalued attributes, which must be pulled out as their own relations. For example, if an Employee type has several phone numbers, a new relation is created with an attribute for the type of phone and an attribute for the phone number itself. A primary key will be chosen for the new relation as a unique attribute, or a combination of unique attributes. In the case where no combination of unique identifiers may be obtained, an artificial key may be created. Foreign keys may be added when the relationships that connect to this entity are converted to relations.

2.2.2 Converting Relationship Types to Relations

In order to convert an ER model to a Relational model, relationship types must also be converted to relations. First, methods for converting relationships to relations will be discussed, then special relationship cases will be addressed. In general, there are three methods for converting relationships to relations. The use of each method is dependent upon the cardinality of the relationship, and the number of entities connected to the relationship. Below is an illustrated summary of each of the methods, and after that is a detailed description of each. The conversion of special cases such as superclass/subclass relationships, shared subclasses, union types and recursive relationships are are discussed after the three main methods of conversion.

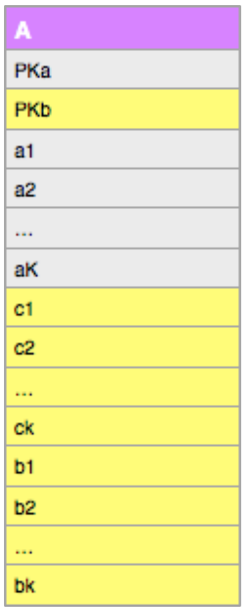


1. foreign key method (Binary relationships 1:1, 1:N, N:1)



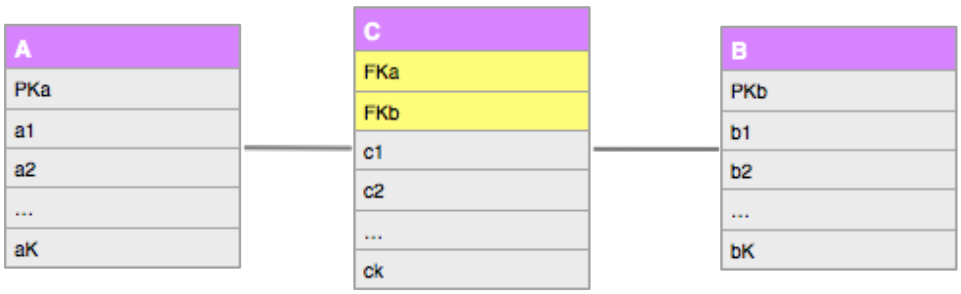
For 1:1 relations, A is the relation that has total participation in the relationship, if any. For 1:N and N:1 relations, A is the relation on the "many" side.

2. merged relation method (Binary relationships 1:1, total participation for both)



Note that the primary key of B is kept when it contains information, but discarded when it was an artificial key

3. merged key method (Binary and N-ary relationships 1:1, 1:N, N:1, M:N)



The first method is commonly used for binary relationships that are not many to many. It may be used for binary relationships with a cardinality of 1:1, 1:N or N:1. It is called the foreign key method. In the foreign key method, the relationship is dissolved, and the two entities it related are then connected by adding the primary key of one as a foreign key to another. The relationship that received the foreign key will also receive any attributes that the relationship originally contained. For 1:1 relationships, it is best to add the foreign key and attributes to the relation that has a total participation. For 1:N and N:1 relationships, the foreign key and relationship attributes must be added to the relation on the “many” side of the relationship, because it will be related to at most 1 other relation.

The second method for converting relationships is called the merged relation method. In the merged relation approach, The relationship is dissolved, and the two entities are then merged together. This method is not commonly used because it may make the resulting relation too big, and it may remove individual meaning when combining the two entities and relationship all into one relation.

The third method for converting relationships is called the merged key method. The merged key method converts the relationship to its own relation. Primary keys of the connected relations are added as foreign keys to the new relation. The merged key method is the only method for converting many to many relationships and n-ary (more than two connected entities) relationships.

Superclass/subclass relationships must also be converted to relations. There are two categories for conversion of superclass/subclass relationships. The first is called multiple-relation options and the second is called single-relation options. Multiple-relation options keep all of the subclass entities as their own relations with their own primary keys. There is the option of keeping the superclass as its own relation, or removing it. It would only be a good idea to remove the superclass if there are little to no attributes contained within it, or it will lead to unnecessary duplication. Single-relation options merge all of the subclasses into their superclass and make it a single relation. There is the option of including a ‘type’ attribute

which specifies the type, or boolean attribute for each type that exists, and setting it true for all of the types it is a part of. Multiple relation options will make data access slightly more difficult. To search for information within each relation would require more queries. Single relation options may lead to the table growing too large with many empty attributes. Shared subclasses (multiple inheritance) may be converted to a relation by the same methods that single inheritance uses. They may be mapped by using the single or multiple relation options. In order to map categories (union types) a new relation is created with an artificial key and a type attribute to specify which superclass the category belongs to. Recursive relationships are where the same entity may participate in the relationship more than once, but as a different role. Since recursive relationships are many to one, they are mapped to a relation by the foreign key method.

2.2.3 Database Constraints

Relations within a database typically correspond or link to each other in some way. This linking places restrictions on the values attributes within the entire relational database may take. These restrictions are put in place to ensure the entire database state is logically consistent, and does not break any rules from the miniworld the database represents, such as business rules and limitations. Such restrictions are called constraints, and may exist for several aspects of the database. The DBMS enforces constraints by checking values upon insertion, deletion, and updating. Some classifications for constraints are described below.

Primary & Unique Key Constraints

A relation is a set of tuples, and therefore each tuple must be distinct. In order to ensure distinctness, a relation is checked for the presence of an attribute, or minimal set of attributes, that must be distinct for every tuple. A minimal attribute set is one in which no more attributes may be removed with the key remaining unique. Any possible minimal attribute set that can uniquely identify each tuple in a relation is termed a “candidate key”. From these possible candidate keys, a single key is selected and constrained to ensure a tuple with duplicate values for the key may never be created. The selected key is called the “primary key”, and the remaining candidate key are labeled “unique keys”.

Entity Integrity Constraint

Since primary keys must be able to uniquely identify tuples, they must all be queryable, and therefore must never be NULL. Being able to set NULL for values in an attribute set implies they are not uniquely identifiable, since any tuple may take on NULL for the value of that attribute. This restriction is known as the Entity Constraint, or Entity Integrity Constraint.

Referential Integrity Constraint

Constraints must also be set to maintain consistency between relations as well. The Referential Constraint, or Referential Integrity Constraint, is designed to ensure any tuples in one relation that refer to tuples in another relation always refer to tuples that exist. To meet this constraint, relations that refer to other relations, or “Referencing Relations”, must contain the primary key of the relation being referenced, or “Referenced Relation”. This referenced relation primary key attribute stored in the referencing relation is called the “Foreign Key”. A foreign key value must be set to an existing primary key value from the referenced relation, or be NULL. This prevents foreign keys from referring to tuples that do not exist.

Check Constraints & Business Rules

Attribute values within tuples are almost always limited to a certain type and range that is specific to the attribute in question. These limitations on the values an attribute can take are known as “Check Constraints, or “Domain Constraints”. Examples of such constraints are character arrays of length 255, or integers within a range of 0 to 500. Check constraints are created as part of a larger set of constraints known as “Business Rules”, which are constraints used to ensure the operating rules of a business are upheld. For example, a vacation_request can be considered a holiday that applies to all employees if it’s requested_by and responded_by attributes are both NULL.

Section 2.3 Convert Entity Relationship Model to Relational Model

This section lists each relationship from the relational model. Each attribute has their domain listed, as well as specifications for any primary, foreign or candidate key constraints.

2.3.1 Relation Schema for Logical Database

Availability	
The availability times will be limited to begin when the earliest shift starts, and end when the latest shift ends. Availability will increment in half hour increments, and must be at least 4 consecutive hours. In the case where an employee has split availability on a given day, a second availability must be created for that day where it does not overlap with the other for the same day. The weekday attribute must be “mon”, “tues”, “wed”, “thurs”, “fri”, “sat”, or “sun”.	
*employee_id	int 0-999, foreign key, primary key
*day	varchar2(16), primary key
*startt	time 0:00 - 24:00, primary key
endt	time 0:00 - 24:00

cannot_work-with	
*employee1_id	int 0-999, foreign key
*employee2_id	int 0-999, foreign key

delivered_by	
*supplier	int 0-999, foreign key, primary key
*ingredient_id	int 0-999, foreign key, primary key
*date_delivered	date January 1, 2000 - December 31, 4712, primary key
quantity	int 0-999

employee	
The birth_date must be long enough ago so that the employee is currently at the age of 15 ½ or older.	
*id	int 0-999, primary key
first_name	varchar2(255)
last_name	varchar2(255)
is_manager	boolean
birth_date	date January 1, 2000 - December 31, 4712
max_hours_per_week	int 0-40
phone_number	long 1000000000-9999999999
clock_number	int 0-99

employment_history	
If multiple employment histories exist for the same employee, the date ranges of the tuples must not intersect.	
*employee_id	int 0-999, foreign key, primary key
*date_employed	date January 1, 2000 - December 31, 4712, primary key
date_unemployed	date January 1, 2000 - December 31, 4712

has_position	
When a position is removed, the date removed is set to that date, but if a position is reacquired, the date_removed is set back to NULL.	
*employee_id	int 0-999, foreign key
*position_id	int 0-999, foreign key
date_acquired	date January 1, 2000 - December 31, 4712
date_removed	date January 1, 2000 - December 31, 4712
is_primary	boolean
is_training	boolean

holiday	
The type of a holiday may be a full or half day.	
*date	date January 1, 2000 - December 31, 4712, primary key
name	varchar2(255)
type	varchar2(255)

ingredient	
*id	int 0-999, primary key
name	varchar2(255)

is_scheduled_for	
An employee is scheduled no more than twice per day (rarely are they scheduled more than once). If two employees are scheduled for the same shift on the same date, that means that one employee is training for the shift with another who has experience with that shift. There will never be more than two employees scheduled for the same shift on the same date.	
*employee_id	int 0-999, foreign key, primary key
*shift_id	int 0-999, foreign key, primary key
*date	date January 1, 2000 - December 31, 4712, primary key

menu_item	
*id	int 0-999, primary key
name	varchar2(255)
type	varchar2(255)
price	number(4,2)
photo	bfile

orders	
*employee_id	int 0-999, foreign key
*ingredient_id	int 0-999, foreign key
date	date January 1, 2000 - December 31, 4712
quantity	int 0-999

position	
The location is limited to “front”, “kitchen”, “both”, or null for none.	
*id	int 0-999, primary key
title	varchar2(255)

requests_vacation	
When responded by is null, the request is considered pending. If the start time is null, the end time must also be null. When both the start and end time are null, the request is for the entire day.	
*requested_by	int 0-999, foreign key, primary key
*responded_by	int 0-999, foreign key, primary key
is_approved	boolean
*start_date_time	timestamp with date January 1, 2000 - December 31, 4712, 0:00-24:00 primary key
end_date_time	timestamp with date January 1, 2000 - December 31, 4712, 0:00-24:00

shift	
*id	int 0-999, primary key
*task_name	varchar2(255), foreign key

sold_in	
*menu_item_id	int 0-999, foreign key
*transaction_id	int 0-999, foreign key
quantity	int 0-999

supplier	
*id	int 0-999, primary key
title	varchar2(255)
delivery_days	varchar2(16)

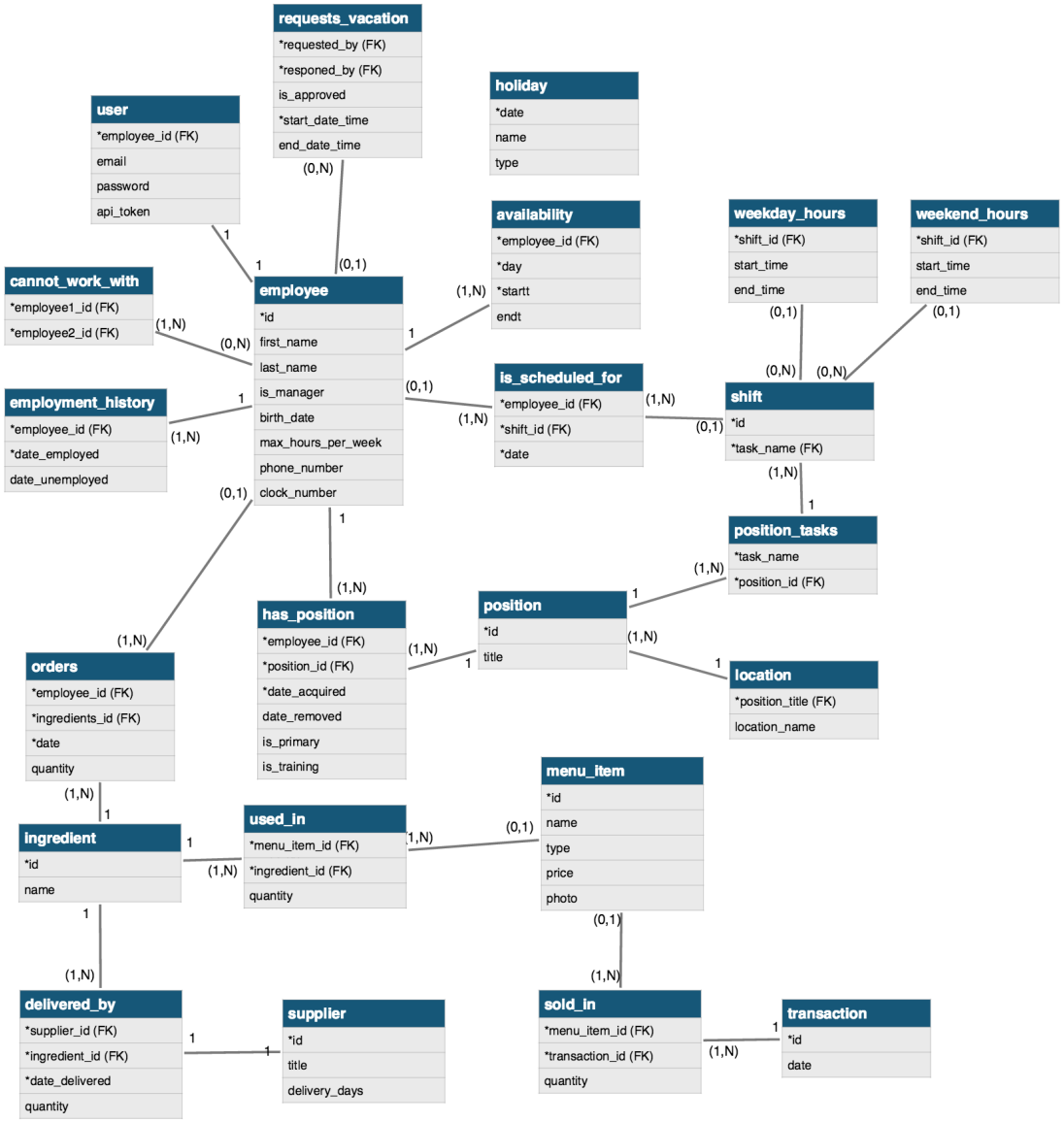
transaction	
*id	int 0-999, primary key
date	date January 1, 2000 - December 31, 4712

used_in	
*menu_item_id	int 0-999, foreign key
*ingredient_id	int 0-999, foreign key
quantity	int 0-999

user	
The password is a hashed value, and not directly stored as it is for security	
*employee_id	int 0-999, primary key
email	varchar2(255), unique
password	varchar2(255)
api_token	varchar2(255), unique

weekday_hours	
Times must be in increments of 30 minutes, and the total hours must be two hours or longer per shift.	
*shift_id	int 0-999, foreign key
start	time 0:00-24:00
end	time 0:00-24:00

weekend_hours	
Times must be in increments of 30 minutes, and the total hours must be two hours or longer per shift.	
*shift_id	int 0-999, foreign key
start_time	time 0:00-24:00
end_time	time 0:00-24:00


requests_vacation
*requested_by (FK)
*responed_by (FK)
is_approved
*start_date_time
end_date_time
holiday
*date
name
type
user
*employee_id (FK)
email
password
api_token
availability
*employee_id (FK)
*day
*startt
endt
weekday_hours
*shift_id (FK)
start_time
end_time
weekend_hours
*shift_id (FK)
start_time
end_time
cannot_work_with
*employee1_id (FK)
*employee2_id (FK)
employee
*id
first_name
last_name
is_manager
birth_date
max_hours_per_week
phone_number
clock_number
is_scheduled_for
*employee_id (FK)
*shift_id (FK)
*date
shift
*id
*task_name (FK)
employment_history
*employee_id (FK)
*date_employed
date_unemployed
position_tasks
*task_name
*position_id (FK)
has_position
*employee_id (FK)
*position_id (FK)
*date_acquired
date_removed
is_primary
is_training
position
*id
title
location
*position_title (FK)
location_name
orders
*employee_id (FK)
*ingredients_id (FK)
*date
quantity
menu_item
*id
name
type
price
photo
used_in
*menu_item_id (FK)
*ingredient_id (FK)
quantity
ingredient
*id
name
delivered_by
*supplier_id (FK)
*ingredient_id (FK)
*date_delivered
quantity
supplier
*id
title
delivery_days
sold_in
*menu_item_id (FK)
*transaction_id (FK)
quantity
transaction
*id
date

2.3.2 Sample Data of Relations

Availability			
*employee_id	* day	*startt	endt
1	mon	6:00	18:00
1	tues	7:00	19:00
1	wed	10:00	20: 00
1	thurs	11:00	21:00
1	fri	6:00	11:00
1	fri	14:00	18:00
1	sun	10:00	20:00
2	mon	6:00	18:00
2	tues	7:00	19:00
2	wed	10:00	20: 00
2	thurs	11:00	21:00
2	fri	6:00	18:00
2	sat	7:00	19:00
2	sun	10:00	20:00
3	mon	6:00	18:00
3	tues	7:00	19:00
3	wed	10:00	20: 00
3	thurs	11:00	21:00
3	fri	6:00	18:00
3	sat	7:00	19:00
3	sun	10:00	20:00
4	mon	6:00	18:00
4	tues	7:00	19:00
4	wed	10:00	20: 00
4	thurs	11:00	21:00
4	fri	6:00	18:00
4	sat	7:00	19:00
4	sun	10:00	20:00
5	mon	6:00	18:00
5	tues	7:00	19:00
5	wed	10:00	20: 00
5	thurs	11:00	21:00
5	fri	6:00	18:00
5	sat	7:00	19:00
5	sun	10:00	20:00
6	mon	6:00	18:00
6	tues	7:00	19:00
6	wed	10:00	20: 00
6	thurs	11:00	21:00
6	fri	6:00	18:00
6	sat	7:00	19:00
6	sun	10:00	20:00
7	mon	6:00	18:00
7	tues	7:00	19:00
7	wed	10:00	20: 00
7	thurs	11:00	21:00
7	fri	6:00	18:00
7	sat	7:00	19:00
7	sun	10:00	20:00

8	mon	6:00	18:00
8	tues	7:00	19:00
9	mon	6:00	18:00
9	tues	7:00	19:00
10	mon	6:00	18:00
10	tues	7:00	19:00
11	mon	6:00	18:00
11	tues	7:00	19:00
12	mon	6:00	18:00
12	tues	7:00	19:00
13	mon	6:00	18:00
13	tues	7:00	19:00
14	mon	6:00	18:00
14	tues	7:00	19:00
15	mon	6:00	18:00
15	tues	7:00	19:00
16	mon	6:00	18:00
16	tues	7:00	19:00
17	mon	6:00	18:00
17	tues	7:00	19:00
18	mon	6:00	18:00
18	tues	7:00	19:00
19	mon	6:00	18:00
19	tues	7:00	19:00
20	mon	6:00	18:00
20	tues	7:00	19:00

delivered_by

*supplier	*ingredient_id	*date_delivered	quantity
1	1	9/30/15	6
1	2	9/30/15	5
1	1	9/29/15	4
1	2	9/29/15	5
1	1	9/28/15	2
1	2	9/28/15	4
1	1	9/27/15	7
1	2	9/27/15	8
2	3	9/28/15	1
2	4	9/28/15	2
2	3	9/27/15	3
2	4	9/27/15	4
2	3	9/26/15	5
2	4	9/26/15	6
3	5	9/27/15	7
3	6	9/27/15	1
3	5	9/26/15	2
3	6	9/26/15	3
3	5	9/25/15	4
3	6	9/25/15	5
4	7	9/26/15	6
4	8	9/26/15	7
4	7	9/25/15	8
4	8	9/25/15	9
4	7	9/24/15	1
4	8	9/24/15	2
5	9	9/25/15	3

5	10	9/25/15	4
5	9	9/24/15	5
5	10	9/24/15	6
5	9	9/23/15	7
5	10	9/23/15	8
6	11	9/24/15	9
6	12	9/24/15	1
6	11	9/23/15	2
6	12	9/23/15	3
6	11	9/22/15	4
6	12	9/22/15	5
7	13	9/23/15	6
7	14	9/23/15	7
7	13	9/22/15	8
7	14	9/22/15	9
7	13	9/21/15	1
7	14	9/21/15	2
8	15	9/22/15	3
8	16	9/22/15	4
8	15	9/21/15	5
8	16	9/21/15	6
8	15	9/20/15	7
8	16	9/20/15	8
8	15	9/19/15	9
9	17	9/21/15	1
9	18	9/21/15	2
9	17	9/20/15	3
10	19	9/21/15	4
10	20	9/21/15	5
10	19	9/20/15	6
10	20	9/20/15	7
10	19	9/19/15	8
10	20	9/19/15	9

employee

*id	first_name	last_name	is_manager	birth_date	max_hours_per_week	phone_number	clock_number
1	John	Doe	False	01/01/85	40	1112223311	129
2	Jane	Doe	True	01/01/85	30	1112223310	128
3	Bob	Doe	False	01/01/85	20	1112223339	127
4	Susan	Doe	False	01/01/85	40	1112223338	126
5	John	Reese	False	01/01/86	30	1112223337	125
6	Jane	Reese	False	01/01/86	20	1112223336	124
7	Bob	Reese	True	01/01/86	40	1112223335	123
8	Susan	Reese	True	01/01/86	40	1112223333	122
9	John	Carter	False	01/01/87	30	1112223331	121
10	Jane	Carter	True	01/01/87	20	1112223331	120
11	Bob	Carter	False	02/13/00	40	1001112220	130
12	Susan	Carter	False	02/13/99	30	1001112221	131
13	John	Jackson	False	02/13/98	20	1001112222	132
14	Jane	Jackson	False	02/13/00	40	1001112223	133
15	Bob	Jackson	False	02/13/99	30	1001112224	134
16	Susan	Jackson	True	02/13/98	20	1001112226	135
17	John	Simpson	False	02/13/00	40	1001112227	136
18	Jane	Simpson	False	02/13/99	30	1001112228	137
19	Bob	Simpson	False	02/13/98	20	1001112229	138
20	Susan	Simpson	False	02/13/97	40	1001112230	139

employment_history		
*employee_id	*date_employed	date_unemployed
1	05/15/14	06/01/14
1	06/30/14	07/31/14
1	08/31/14	09/30/14
1	10/31/14	NULL
2	08/01/15	01/01/16
2	10/01/15	NULL
3	01/01/13	NULL
4	02/01/13	NULL
5	03/01/13	NULL
6	04/01/13	NULL
7	05/01/13	NULL
8	06/01/13	NULL
9	07/01/13	NULL
10	08/01/13	NULL
11	09/01/13	NULL
12	10/01/13	NULL
13	11/01/13	NULL
14	12/01/13	NULL
15	01/01/14	NULL
16	02/01/14	NULL
17	03/01/14	NULL
18	04/01/14	NULL
19	05/01/14	NULL
20	06/01/14	NULL

has_position					
*employee_id	*position_id	*date_acquired	date_removed	is_primary	is_training
1	1	05/15/14	NULL	True	False
1	2	06/30/14	07/31/14	False	False
1	3	08/31/14	09/30/14	False	False
2	5	08/01/15	01/01/16	False	False
2	6	10/01/15	NULL	True	True
2	10	10/01/15	NULL	False	True
3	8	01/01/13	NULL	True	False
3	9	02/01/13	NULL	False	False
3	1	03/01/13	NULL	False	True
4	2	02/01/13	NULL	True	False
4	3	04/01/13	NULL	False	False
4	4	05/01/13	NULL	False	False
5	3	03/01/13	NULL	True	False
5	6	07/01/13	NULL	False	False
5	7	08/01/13	NULL	False	False
6	8	04/01/13	NULL	True	False
6	9	10/01/15	NULL	False	True
6	1	06/01/13	NULL	False	False
7	10	05/01/13	NULL	True	False
7	3	06/01/13	NULL	False	False
7	4	07.01/13	NULL	False	False
8	10	06/01/13	NULL	True	False
8	6	07/01/13	NULL	False	False
8	7	10/02/15	NULL	False	True
9	8	07/01/13	NULL	True	False
9	9	08/01/13	NULL	False	False
9	1	09/02/13	NULL	False	False

cannot_work_with	
*employee1_id	*employee2_id
1	2
1	3
1	4
2	5
2	6
5	3
6	7
7	9
8	10
9	12

10	2	08/01/13	NULL	True	False
10	3	09/01/14	NULL	False	False
10	10	10/01/14	NULL	False	False
11	2	09/01/13	NULL	True	False
11	6	10/01/13	NULL	False	False
11	7	11/01/13	NULL	False	False
12	8	10/01/13	NULL	True	False
12	9	11/01/13	NULL	False	False
12	1	10/03/15	NULL	False	True
13	2	11/01/13	NULL	True	False
13	3	12/01/13	NULL	False	False
13	4	01/01/14	NULL	False	False
14	3	12/01/13	NULL	True	False
14	6	02/01/14	NULL	False	False
14	7	03/01/14	NULL	False	False
15	8	01/01/14	NULL	True	False
15	9	02/01/14	NULL	False	False
15	1	03/01/14	NULL	False	False
16	2	02/01/14	NULL	True	False
16	3	03/01/14	NULL	False	False
16	10	04/01/14	NULL	False	False
17	1	03/01/14	NULL	True	False

17	6	04/01/14	NULL	False	False
17	7	05/01/15	NULL	False	False
18	8	04/01/14	NULL	True	False
18	9	10/14/15	NULL	False	True
18	1	10/04/15	NULL	False	True
19	2	05/01/14	NULL	True	False
19	3	06/01/14	NULL	False	False
19	4	07/01/14	NULL	False	False
20	2	06/01/14	NULL	True	False
20	3	07/01/14	NULL	False	False
20	4	08/01/14	NULL	False	False

holiday		
*date	name	type
12/25	Christmas	full
12/24	Christmas Eve	half
12/31	New Years Eve	half
01/01	New Years Day	half
10/31	Halloween	half
11/27	Thanks Giving	full
06/07	Easter	full

ingredient	
*id	name
1	Potato
2	Tomato
3	Broccoli
4	Corn
5	Rice
6	Jalapeno
7	Yellow Onion
8	Red Onion
9	Flour
10	Chicken
11	Pork
12	Beef
13	Bacon
14	Turkey
15	Tuna
16	Cheddar Cheese
17	Swiss Cheese
18	Provolone Cheese
19	Yellow Mustard
20	Honey Mustard

is_scheduled_for		
*employee_id	*shift_id	*date
1	1	10/20/15
2	10	10/20/15
3	3	10/20/15
4	17	10/20/15

5	6	10/20/15
6	3	10/28/15
7	7	10/20/15
8	10	10/20/15
9	6	10/28/15
10	2	10/21/15
11	19	10/21/15
12	17	10/21/15
13	6	10/21/15
14	8	10/21/15
15	15	10/21/15
16	10	10/21/15
17	13	10/21/15
18	11	10/21/15
19	6	10/22/15
20	8	10/22/15
1	2	10/22/15
2	10	10/22/15
3	5	10/22/15
4	12	10/22/15
5	7	10/22/15
6	4	10/23/15
7	8	10/23/15
8	20	10/23/15
9	13	10/23/15
10	9	10/23/15
11	2	10/23/15
12	3	10/23/15
13	7	10/24/15
14	9	10/24/15
15	14	10/24/15
16	10	10/24/15
17	13	10/24/15
18	11	10/24/15
19	6	10/25/15
20	7	10/25/15
1	9	10/25/15
2	20	10/25/15
3	15	10/25/15
4	18	10/25/15
5	8	10/26/15
6	5	10/26/15
7	16	10/26/15
8	20	10/26/15
9	15	10/26/15
10	10	10/26/15
11	2	10/26/15
12	18	10/27/15
13	16	10/27/15
14	7	10/27/15
15	13	10/27/15
16	10	10/27/15
17	11	10/27/15
18	5	10/27/15
19	6	10/27/15
20	8	10/27/15

menu_item

*id	name	type	price	photo
1	BBQ Chicken	Sandwich	7.95	(png file)
2	California Chicken Breast	Sandwich	7.95	(png file)
3	Tuna Melt	Sandwich	7.95	(png file)
4	Turkey Bacon Melt	Sandwich	7.95	(png file)
5	Classic Ruben	Sandwich	7.95	(png file)
6	California Pastrami	Sandwich	7.95	(png file)
7	Country Potato	Cream Soup	5.95	(png file)
8	Tomato Bisque	Cream Soup	5.95	(png file)
9	Brocoli & Cheese	Cream Soup	5.95	(png file)
10	Corn Chowder	Cream Soup	5.95	(png file)

orders			
*employee_id	*ingredient_id	*date	quantity
2	1	10/17/15	5
7	2	10/17/15	6
8	3	10/17/15	7
10	4	10/17/15	8
16	5	10/17/15	9
2	6	10/18/15	10
7	7	10/18/15	11
8	8	10/18/15	12
10	9	10/18/15	5
16	10	10/18/15	6
2	11	10/19/15	7
7	12	10/19/15	8
8	13	10/19/15	9
10	14	10/19/15	10
16	15	10/19/15	11
2	16	10/20/15	12
7	17	10/20/15	5
8	18	10/20/15	6
10	19	10/20/15	7
16	20	10/20/15	8
2	1	10/20/15	9
7	2	10/20/15	10
8	3	10/20/15	11
10	4	10/20/15	12
16	5	10/20/15	5
2	6	10/21/15	6
7	7	10/21/15	7
8	8	10/21/15	8
10	9	10/21/15	9
16	10	10/21/15	10
2	11	10/22/15	11
7	12	10/22/15	12
8	13	10/22/15	5
10	14	10/22/15	6
16	15	10/22/15	7
2	16	10/22/15	8

7	17	10/23/15	9
8	18	10/23/15	10
10	19	10/23/15	11
16	20	10/23/15	12
2	1	10/23/15	5
7	2	10/23/15	6
8	3	10/23/15	7
10	4	10/23/15	8
16	5	10/24/15	9
2	6	10/24/15	10
7	7	10/24/15	11
8	8	10/24/15	12
10	9	10/24/15	5
16	10	10/24/15	6
2	11	10/25/15	7
7	12	10/25/15	8
8	13	10/25/15	9
10	14	10/25/15	10
16	15	10/25/15	11
2	16	10/26/15	12
7	17	10/26/15	5
8	18	10/26/15	6
10	19	10/26/15	7
16	20	10/26/15	8

position	
*id	title
1	Cashier
2	Janitor
3	Kitchen
4	Cold Prep
5	Pos1
6	Pos2
7	Pos3
8	Pos4
9	Kitchen Supervisor
10	Supervisor

position-tasks	
position_id	*task_name
1	Register1
1	Register2
2	Opening Janitor
2	Closing Janitor
3	Cookies
3	Soups
4	Meat
4	Cheese
5	Kitchen Supervisor
5	task A
6	task B
6	task C
7	task D
7	task E

8	task F
8	task G
9	task H
9	task I
10	Open Store
10	Close Store

requests_vacation						
*requested_by	*responded_by	is_approved	*start_date_time		end_date_time	
1	7	True	01/02/15	00:00	01/12/15	24:00
2	8	True	01/02/15	00:00	01/12/15	24:00
3	10	False	01/02/15	00:00	01/12/15	24:00
4	16	False	01/02/15	00:00	01/12/15	24:00
5	7	False	01/03/15	00:00	01/13/15	24:00
6	8	False	01/03/15	00:00	01/13/15	24:00
9	10	True	01/03/15	00:00	01/13/15	24:00
11	16	False	01/03/15	00:00	01/13/15	24:00
12	7	False	01/04/15	00:00	01/14/15	24:00
13	8	False	01/04/15	00:00	01/14/15	24:00
14	10	False	01/04/15	00:00	01/14/15	24:00
15	16	True	01/04/15	00:00	01/14/15	24:00
17	7	False	01/05/15	00:00	01/15/15	24:00
18	8	False	01/05/15	00:00	01/15/15	24:00
19	10	False	01/05/15	00:00	01/15/15	24:00
20	16	False	01/05/15	00:00	01/15/15	24:00
1	7	True	03/05/15	00:00	03/15/15	24:00
2	8	False	03/05/15	00:00	03/15/15	24:00
3	10	False	03/05/15	00:00	03/15/15	24:00
4	16	True	03/05/15	00:00	03/15/15	24:00
5	7	False	03/06/15	00:00	03/16/15	24:00
6	8	False	03/06/15	00:00	03/16/15	24:00
9	10	False	03/06/15	00:00	03/16/15	24:00
11	16	False	03/06/15	00:00	03/16/15	24:00
12	7	True	03/06/15	00:00	03/16/15	24:00
13	8	False	03/07/15	00:00	03/17/15	24:00
14	10	False	03/07/15	00:00	03/17/15	24:00
15	16	True	03/07/15	00:00	03/17/15	24:00
17	7	True	03/07/15	00:00	03/17/15	24:00
18	8	False	03/07/15	00:00	03/17/15	24:00
19	10	False	03/07/15	00:00	03/17/15	24:00
20	16	False	03/07/15	00:00	03/17/15	24:00
1	7	True	07/01/15	00:00	07/11/15	24:00
2	8	False	07/01/15	00:00	07/11/15	24:00

3	10	False	07/01/15	00:00	07/11/15	24:00
4	16	False	07/01/15	00:00	07/11/15	24:00
5	7	False	07/01/15	00:00	07/11/15	24:00
6	8	True	07/01/15	00:00	07/11/15	24:00
9	10	False	07/02/15	00:00	07/12/15	24:00
11	16	False	07/02/15	00:00	07/12/15	24:00
12	7	False	07/02/15	00:00	07/12/15	24:00
13	8	False	07/02/15	00:00	07/12/15	24:00
14	10	True	07/02/15	00:00	07/12/15	24:00
15	16	True	07/02/15	00:00	07/12/15	24:00
17	7	False	07/03/15	00:00	07/13/15	24:00
18	8	False	07/03/15	00:00	07/13/15	24:00
19	10	False	07/03/15	00:00	07/13/15	24:00
20	16	False	07/03/15	00:00	07/13/15	24:00
1	NULL	NULL	11/12/15	00:00	11/22/15	24:00
2	NULL	NULL	11/12/15	00:00	11/22/15	24:00
3	NULL	NULL	11/12/15	00:00	11/22/15	24:00
4	NULL	NULL	11/12/15	00:00	11/22/15	24:00
5	NULL	NULL	11/13/15	00:00	11/23/15	24:00
6	NULL	NULL	11/13/15	00:00	11/23/15	24:00
9	NULL	NULL	11/13/15	00:00	11/23/15	24:00
11	NULL	NULL	11/13/15	00:00	11/23/15	24:00
NULL	NULL	True	09/07/15	00:00	09/07/15	24:00
NULL	NULL	True	11/26/15	00:00	11/27/15	24:00
NULL	NULL	True	12/25/15	00:00	12/25/15	24:00
NULL	NULL	True	12/31/15	12:00	01/01/15	24:00

shift	
*id	task_name
1	Open Support
2	Opening Janitor
3	Register1
4	Register2
5	Mid Morning
6	Cookies
7	Hot Prep
8	Salads
9	Soups
10	Opening Supervisor
11	Closing Cashier
12	Closing Janitor
13	Dressings
14	Tomatoes
15	Cut Cakes
16	Meat
17	Cheese
18	Bakery
19	Sysco

20	Closing Supervisor

sold_in		
*menu_item_id	*transaction_id	quantity
1	1	1
2	1	2
3	1	3
4	1	1
5	1	2
6	1	3
7	2	1
8	2	2
9	2	3
10	2	1
1	2	2
2	2	3
3	3	1
4	3	2
5	3	3
6	3	1
7	3	2
8	3	3
9	4	1
10	4	2
1	4	3
2	4	1
3	4	2
4	4	3
5	5	1
6	5	2
7	5	3
8	5	1
9	5	2
10	5	3
1	6	1
2	6	2
3	6	3
4	6	1
5	6	2
6	6	3
7	7	1
8	7	2
9	7	3
10	7	1
1	7	2
2	7	3
3	8	1
4	8	2
5	8	3
6	8	1
7	8	2
8	8	3
9	9	1
10	9	2
1	9	3
2	9	1
3	9	2

4	9	3
5	10	1
6	10	2
7	10	3
8	10	1
9	10	2
10	10	3

supplier		
*id	title	delivery_days
1	Pepsi	m
2	Sysco	t r s
3	Alpha	m t w r f s
4	Boar's Head	m
5	Gellaso's	m w r
6	Pyreneese	m t w r f s
7	Beta	f s
8	Kappa	m w s
9	Gamma	t f
10	Delta	w r

transaction	
*id	date
1	10/25/15
2	10/25/15
3	10/25/15
4	10/26/15
5	10/26/15
6	10/26/15
7	10/26/15
8	10/27/15
9	10/27/15
10	10/27/15

used_in		
*menu_item_id	*ingredient_id	quantity
1	2	2
1	8	2
1	9	2
1	10	2
1	18	1
1	19	1
2	2	2
2	8	2
2	9	2
2	10	2
2	17	1
2	20	1
3	2	2
3	8	2
3	9	2
3	15	2
3	17	1

3	20	1
4	2	2
4	8	2
4	9	2
4	13	2
4	14	1
4	18	1
5	2	2
5	8	2
5	9	2
5	12	2
5	16	1
5	19	1
6	2	2
6	8	2
6	9	2
6	11	2
6	16	1
6	19	1
7	1	2
7	6	1
7	7	1
7	10	1
7	12	2
7	16	2
8	2	2
8	6	1
8	7	1
8	10	1
8	12	2
8	16	2
9	3	2
9	6	1
9	7	1
9	10	1
9	12	2
9	16	2
10	4	2
10	6	1
10	7	1
10	10	1
10	12	2
10	16	2

user			
*employee_id	email	password	api_token
1	emp1@server.com	12345	token_string_1
2	emp2@server.com	pASsWord	token_string_2
3	emp3@server.com	12345	token_string_3
4	emp4@server.com	pASsWord	token_string_4
5	emp5@server.com	12345	token_string_5
6	emp6@server.com	pASsWord	token_string_6
7	emp7@server.com	12345	token_string_7
8	emp8@server.com	pASsWord	token_string_8

9	emp9@server.com	12345	token_string_9
10	emp10@server.com	pASsWord	token_string_10
11	emp11@server.com	12345	token_string_11
12	emp12@server.com	pASsWord	token_string_12
13	emp13@server.com	12345	token_string_13
14	emp14@server.com	pASsWord	token_string_14
15	emp15@server.com	12345	token_string_15
16	emp16@server.com	pASsWord	token_string_16
17	emp17@server.com	12345	token_string_17
18	emp18@server.com	pASsWord	token_string_18
19	emp19@server.com	12345	token_string_19
20	emp20@server.com	pASsWord	token_string_20

weekday_hours

*shift_id	start	end
1	6:00	14:00
2	7:00	15:00
3	8:00	15:00
4	8:00	15:30
5	9:00	14:00
6	11:00	16:00
7	15:00	20:30
8	15:30	20:30
9	15:00	21:00
10	16:00	21:00

weekend_hours

*shift_id	start	end
1	6:00	14:00
2	7:00	15:00
3	8:00	15:00
4	8:00	15:30
5	9:00	14:00
6	11:00	16:00
7	15:00	20:30
8	15:30	20:30
9	15:00	21:00
10	16:00	21:00

Section 2.4 Sample Queries

This section lists ten sample queries, and mathematically models them three different ways. First, they are modeled in relational algebra, next they are modeled in tuple relational calculus and then domain relational calculus. Each section briefly describes their respective languages.

2.4.1 Design of Queries

1. Find all employees who have been scheduled to work as a closing janitor.
2. Find all employees who can work the tasked named register1 on Mondays. Include employees still training for the task.
3. Find all cashiers available from 14:00-21:00 on all weekdays
4. Find employees who can work every position in the kitchen, where they are not still training for any of them
5. find all supervisors who can work the earliest supervisor shift on Saturday or Sunday.
6. List the employees who have the second longest request for vacation
7. List the possible shifts that “John Doe” can work, given his availability and training
8. Find all employees who are trained in exactly 2 different positions, who can work with all other employees
9. Find sold menu items that used ingredients ordered between 01/01/15 and 01/31/15.
10. Find all employees who have been scheduled exactly once for a cashier shift

2.4.2 Relational Algebra Expressions for Queries

Relational algebra is a procedural mathematical language that uses a set of operations on relations to produce a resulting relation. A procedural language is where there is a specified order of operations in the equation, and certain operations take precedence over others. Query languages such as SQL are derived from relational algebra. There are two types of operations. the first type are operations derived from mathematical set theory. The second type are designed for relational database operations.

1. Find all employees who have been scheduled to work as a closing janitor.

Referenced Relations

shift(*sid, *pid, tname) ← shift(*id,*position_id, task_name)

is_scheduled_for(*eid, *sid, date) ← is_scheduled_for(*employee_id, *shift_id, date)

employee(*eid, fname, lname, manager, bday, maxhrswk, phone, clock) ← employee(*id, first_name, last_name, is_manager, birth_date, max_hours_per_week, phone_number, clock_number)

employment_history(*eid, emp, unemp) ← employment_history(*employee_id, date_employed, date_unemployed)

$$CJShifts \leftarrow \sigma_{(s.tname=\text{"closing janitor"})}(shift)$$

Intermediate Result: CJShifts(sid, pid, tname="closing janitor")

$$EmplSchedlAsCJ \leftarrow \pi_{(i.eid)}(is\_scheduled\_for * CJShifts)$$

Intermediate Result: EmplySchedlAsCJ(eid)

$$CJEmployees \leftarrow \pi_{(e.eid)}(employee) * EmplSchedlAsCJ * \pi_{(h.id)}(\sigma_{(h.unemp=null)}(employment\_history))$$

Final Result: CJEmployees(eid)

The ids of all active employees that have ever been scheduled to work the task "closing janitor".

2. Find all employees who can work the task named register1 on Mondays. Include employees still training for the task.

Referenced Relations

shift(*sid, *pid, tname) ← shift(*id,*position_id, task_name)

weekday_hours (sid, start, end) ← weekday_hours(*shift_id, start, end)

has_position(*eid, *pid, acq, rem, prim, train) ← has_position(*employee_id, *position_id, date_acquired, date_removed, is_primary, is_training)

availability(*eid, day, startt, endt) ← availability(*employee_id, day, startt, endt)

employment_history(*eid, emp, unemp) ← employment_history(*employee_id, date_employed, date_unemployed)

r1weekdayHours ← $(\sigma_{(s.tname=\text{"register1"})}(s(\text{shift})) * w(\text{weekday\_hours})$

Intermediate Result: $r1weekdayHours_{(sid, pid, tname, start, end)}$

r1trained ← $\pi_{(h.eid)}(\sigma_{(h.removed=null \wedge h.train=false)}(\text{r1weekdayHours} * w(\text{has\_position})))$

Intermediate Result: $r1trained_{(eid)}$

empMonAvail ← $\pi_{(a.eid,\ a.startt,\ a.endt)}(\sigma_{(a.day=\text{"mon"})}a(\text{availability}))$

Intermediate Result: $empMonAvail_{(eid,\ startt,\ endt)}$

empMonR1 ← empMonAvail * r1trained

Intermediate Result: $empMonR1_{(eid,\ startt,\ endt)}$

result ← $\pi_{(e.eid)}(\sigma_{(r.start \geq e.startt \wedge r.end \leq e.endt)}(\text{r1weekdayHours} \times r(\text{empMonR1})))$

Final Result: $result_{(eid)}$

The ids of employees who are trained to work register1, and have availability to work register1

3. Find all cashiers available from 14:00 to 21:00 on all weekdays

Referenced Relations

position(*pid, title) ← position(*id, title)

has_position(*eid, *pid, acq, rem, prim, train) ← has_position(*employee_id, *position_id, date_acquired, date_removed, is_primary, is_training)

availability(*eid, day, startt, endt) ← availability(*employee_id, day, startt, endt)

employment_history(*eid, emp, unemp) ← employment_history(*employee_id, date_employed, date_unemployed)

$$\text{CashierID} \leftarrow \pi_{pid}(\sigma_{(p.title=\text{"cashier"})}(\overset{p}{\text{position}}))$$

Intermediate Result: $\text{CashierID}_{(pid)}$

$$\text{Cashiers} \leftarrow \pi_{eid}(\sigma_{(h.rem=null)}(\overset{h}{\text{has\_position}}) * \overset{c}{\text{CashierID}})$$

Intermediate Result: $\text{Cashiers}_{(eid)}$

$$\text{Monday} \leftarrow \pi_{c.eid}(\text{Cashiers} \bowtie_{(a.day=\text{"mon"}\ a.startt\leq 14:00 \wedge a.endt\geq 21:00)} \overset{a}{\text{availability}})$$

$$\text{Tuesday} \leftarrow \pi_{c.eid}(\text{Cashiers} \bowtie_{(a.day=\text{"tues"}\ a.startt\leq 14:00 \wedge a.endt\geq 21:00)} \text{availability})$$

$$\text{Wednesday} \leftarrow \pi_{c.eid}(\text{Cashiers} \bowtie_{(a.day=\text{"wed"}\ a.startt\leq 14:00 \wedge a.endt\geq 21:00)} \text{availability})$$

$$\text{Thursday} \leftarrow \pi_{c.eid}(\text{Cashiers} \bowtie_{(a.day=\text{"thurs"}\ a.startt\leq 14:00 \wedge a.endt\geq 21:00)} \text{availability})$$

$$\text{Friday} \leftarrow \pi_{c.eid}(\text{Cashiers} \bowtie_{(a.day=\text{"fri"}\ a.startt\leq 14:00 \wedge a.endt\geq 21:00)} \text{availability})$$

$$\text{result} \leftarrow \pi_{(\overset{m}{m}.eid)}(\text{Monday} * \text{Tuesday} * \text{Wednesday} * \text{Thursday} * \text{Friday})$$

Final Result: $\text{result}_{(eid)}$

The ids of employees who are available to work anytime between 14:00 and 21:00 on weekdays, and are trained as or are training as cashiers.

4. Find employees who can work every position in the kitchen, where they are not still training for any of them

Referenced Relations

position(*sid, title) ← position(*id, title)

has_position(*eid, *pid, acq, rem, prim, train) ← has_position(*employee_id, *position_id, date_acquired, date_removed, is_primary, is_training)

$$result \leftarrow (\pi_{h.pid,\ h.eid}(\sigma_{h.train=false,\ h.rem=null}(has\_position\ h))) \div (\pi_{p.pid}(\sigma_{p.location=\text{"kitchen"}}(position\ p)))$$

Final Result: $result_{(eid)}$

Employees who can work all positions in the kitchen.

5. find all supervisors who can work the earliest supervisor shift on Saturday or Sunday.

Referenced Relations

position(*pid, title) ← position(*id, title)

shift(*sid, *pid, tname) ← shift(*id,*position_id, task_name)

weekend_hours (sid, start, end) ← weekend_hours(*shift_id, start, end)

has_position(*eid, *pid, acq, rem, prim, train) ← has_position(*employee_id, *position_id, date_acquired, date_removed,is_primary, is_training)

employment_history(*eid, emp, unemp) ← employment_history(*employee_id, date_employed, date_unemployed)

availability(*eid, day, startt, endt) ← availability(*employee_id, day, startt, endt)

$$superShift \leftarrow ((\sigma_{(p.title=\text{"supervisor"})}(position^{p}) * (shift^{s})) * weekend\_hours^{w})$$

Intermediate Result: $superShift_{(pid, sid, title, name, start, end)}$

$$earlyShift \leftarrow \pi_{(s.start,\ s.end)}(superShift^{s} - \pi_{(s1.*)}(\sigma_{(s1.start\ >\ s2.start)}(superShift^{s1} \times superShift^{s2})))$$

Intermediate Result: $earlyShift_{(start, end)}$

$$supervisors \leftarrow \pi_{(h.eid)}(has\_position^{s1} * superShift^{s2})$$

Intermediate Result: $supervisors_{(eid)}$

$$result \leftarrow supervisors * \sigma_{((a.day=\text{"sat"}\ \vee\ a.day=\text{"sun"})\ \wedge\ a.startt \geq s.start\ \wedge\ a.endt \leq s.end)}availability^{a}$$

Final Result: $result_{(eid)}$

Employees who have the position as supervisor.

6. List any employees who have worked for the company and has the second longest request for vacation.

Referenced Relations

requests_vacation(*rid, *requested, *responded, approved, startD, startT, endD, endT) ←
requests_vacation(*id, *requested_by,
*responded_by, is_approved, start_date, start_time, end_date, end_time)

vacNotGreatest ← $\pi_{(r1.*)}(\sigma_{(r1.endD-r1.startD < r2.endD-r2.startD \wedge r1.rid != r2.rid)}$
$(r1(requests\_vacation) \times r2(requests\_vacation))$
)

Intermediate Result: vacNotGreatest(id, requested_by, responded_by,is_approved, start_date, start_time, end_date, end_time)

v

result ← $\pi_{v.requested}$(vacNotGreatest - $\pi_{v1.*}(\sigma_{(v1.endD-v1.startD < v2.endD-v2.startD \wedge v1.rid !=v2.rid)}$
$(v1(vacNotGreatest) \times v2(vacNotGreatest))$
)
)

Final Result: result(requested_by)
The id of the employee who requested the second longest vacation time.

7. List the possible shifts that “John Doe” can work, given his availability and training

Referenced Relations

shift(*sid, *pid, tname) ← shift(*id,*position_id, task_name)

employee(*eid, fname, lname, manager, bday, maxhrswk, phone, clock) ← employee(*id, first_name, last_name, is_manager, birth_date, max_hours_per_week, phone_number, clock_number)

has_position(*eid, *pid, acq, rem, prim, train) ← has_position(*employee_id, *position_id, date_acquired, date_removed, is_primary, is_training)

availability(*eid, day, startt, endt) ← availability(*employee_id, day, startt, endt)

weekday_hours (sid, wdstart, wdend) ← weekday_hours(*shift_id, wdstart, wdend)

weekend_hours (sid, westart, weend) ← weekend_hours(*shift_id, westart, weend)

$JD \leftarrow \pi_{(eid)}(\sigma^{e}_{(e.fname=\text{”John”} \wedge e.lname=\text{”Doe”})}(employee))$

Intermediate Result: $JD_{(eid)}$

$JDPositions \leftarrow \pi_{pid}(\sigma^{h}_{(h.train=false \wedge h.rem=null)}(has\_position) * JD)$

Intermediate Result: $JDPositions_{(pid)}$

JDAvail ← availability * JD

Intermediate Result: $JDAvail_{(*eid, day, startt, endt)}$

$JDPositionShifts \leftarrow \pi^{s}_{(s.sid)}(shift * JDPositions * weekday\_hours * weekend\_hours)$

Intermediate Result: $JDPositionShifts_{(sid)}$

$result \leftarrow \pi_{(j.sid)}(JDAvail \bowtie^{a}_{(p.wdstart \geq a.startt \wedge p.wdend \leq a.endt)} JDPositionShifts^{p})$

Final Result: $result_{(sid)}$

The shifts John Doe may be scheduled for, filtered by his availability and training.

8. Find all employees who are trained in at least 2 different positions, who can work with all other employees

Referenced Relations

has_position(*eid, *pid, acq, rem, prim, train) ← has_position(*employee_id, *position_id, date_acquired, date_removed, is_primary, is_training)

employee(*eid, fname, lname, manager, bday, maxhrswk, phone, clock) ← employee(*id, first_name, last_name, is_manager, birth_date, max_hours_per_week, phone_number, clock_number)

cannot_work_with(*e1, *e2) ← cannot_work_with(*employee1_id, *employee2_id)

employment_history(*eid, emp, unemp) ← employment_history(*employee_id, date_employed, date_unemployed)

$$twoPos \leftarrow \pi_{(h1.id)}(\sigma_{(h1.eid = h2.eid \wedge h1.pid != h2.pid)}(has\_position^{h1} \times has\_position^{h2}))$$

Intermediate Result: $twoPos_{(eid)}$

$$goodEmp \leftarrow \pi_{(e.eid)}(\sigma_{(e.eid != c.e1 \wedge e.eid != c.e2)}(employee^{e} \times cannot\_work\_with^{c}))$$

Intermediate Result: $goodEmp_{(eid)}$

result ← (goodEmp * twoPos)

Final Result: $result_{(eid)}$

The employees who can work exactly two positions and have no restrictions on who they may work with.

9. Find sold menu items that used ingredients ordered between 01/01/15 and 01/31/15.

Referenced Relations

orders(*eid, *iid, date, quantity) ← orders(*employee_id, *ingredient_id, date, quantity)

used_in(*mid, *iid, quantity) ← used_in(*menu_item_id, *ingredient_id, quantity)

sold_in(*mid, *tid, quantity) ← sold_int(*menu_item_id, *transaction_id, quantity)

$$\text{JanIngredients} \leftarrow \pi_{(o.iid)}(\sigma_{(date \geq 01/01/15 \wedge date \leq 01/31/15)}(\rho_{o}(orders)))$$

Intermediate Result: JanIngredients(mid)

$$result \leftarrow \pi_{(u.mid)}(\text{JanIngredients} * \pi_{(u.mid,\ u.iid)}(\rho_{u}(used\_in)) * \pi_{(s.mid)}(\rho_{s}(sold\_in)))$$

Final Result: result(mid)

The menu items that were sold and used ingredients ordered during January, 2015.

10. Find all employees who have been scheduled exactly once for a cashier shift

Referenced Relations

position(*pid, title) ← position(*id, title)

shift(*sid, *pid, tname) ← shift(*id,*position_id, task_name)

is_scheduled_for(*eid, *sid, date) ← is_scheduled_for(*employee_id, *shift_id, date)

employment_history(*eid, emp, unemp) ← employment_history(*employee_id, date_employed, date_unemployed)

$$\text{schCash} \leftarrow \pi_{(p.sid,i.date)}(\sigma_{(p.title=\text{"cashier"})}(\rho_{p}(position) * \rho_{s}(shift)) * \rho_{i}(is\_scheduled\_for))$$

Intermediate Result: schCash(eid, sid, date)

$$result \leftarrow \pi_{(s.eid)}(\rho_{s}(\text{schCash}) - \pi_{(s1.eid)}(\sigma_{(s1.date!=s2.date\ \vee\ s1.sid!=s2.sid\ \wedge\ s1.eid=s2.eid)}(\rho_{s1}(\text{schCash}) \times \rho_{s2}(\text{schCash}))))$$

Final Result: result(eid)

employees scheduled exactly once for a cashier shift

2.4.3 Tuple Relational Calculus Expressions for Queries

While relational algebra is a procedural language, relational calculus is a declarative language that based in mathematical logic. Relational algebra specifies how to get the resulting relation, while relational calculus specifies what the result should have.

In tuple relational calculus the variables are related to tuples. When a variable is declared, it is specified of which relation that it belongs to. A tuple relational calculus equation is in the form { t | cond(t) } where t is the free variable that takes on all values that it is qualified with, and the cond(t) is a true or false condition statement where the result set will include t when the condition statement is true. Several conditions may be connected by logical expressions 'and' (∧), 'or' (∨) and 'implies' (→), and conditions may be nested within other conditions.

The variables are quantified as universal or existential. A universal quantification (∀) means that the variable will be returned if the condition is true for all values specified by the condition. The universal quantifier by definition takes on all values that exist in the universe, so it is common practice to limit any universally quantified variables to those that exist in the database. An existential quantification (∃) means that the variable will be returned if the condition exists for some value. The expression operates, and collects all values where cond(t) is returned true, and adds them to the resulting relation.

1. Find all employees who have been scheduled to work as a closing janitor.

Referenced Relations

shift(*sid, pid, tname) ← shift(*id,*position_id, task_name)

is_scheduled_for(*eid, *sid, date) ← is_scheduled_for(*employee_id, *shift_id, date)

employee(*eid, fname, lname, manager, bday, maxhrswk, phone, clock) ← employee(*id, first_name, last_name, is_manager, birth_date, max_hours_per_week, phone_number, clock_number)

employment_history(*eid, emp, unemp) ← employment_history(*employee_id, date_employed, date_unemployed)

$$\{ e \mid employee(e) \wedge (\exists e)(\exists i_s)(\exists e_h)(shift(s) \wedge is\_scheduled\_for(i_s) \wedge employment\_history(e_h) \wedge s.tname = \text{“closing janitor”} \wedge s.sid = i_s.sid \wedge i_s.eid = e.eid \wedge e.eid = e_h.eid \wedge e_h.unemp = null) \}$$

2. Find all employees who can work the task named register1 on Mondays. Includes employees still training for the task.

Referenced Relations

shift(sid, pid, tname) ← shift(*id,position_id, task_name)

employment_history(hid, empl, unemp) ← employment_history(*id, date_employed, date_unemployed)

weekday_hours(sid, start, end) ← weekday_hours(*shift_id, start, end)

has_position(*eid, *pid, acq, rem, prim, train) ← has_position(*employee_id, *position_id, date_acquired, date_removed, is_primary, is_training)

availability(*eid, day, startt, endt) ← availability(*employee_id, day, startt, endt)

$$\{ e \mid employee(e) \wedge (\exists h)(\exists hp)(\exists s)(\exists w)(\exists a)(employement\_history(h) \wedge has\_position(hp) \wedge shift(s) \wedge weekday\_hours(w) \wedge availability(a) \wedge h.eid=e.eid \wedge h.unemp=null \wedge hp.eid=e.eid \wedge hp.removed=null \wedge s.pid=hp.pid \wedge s.tname=\text{”register1”} \wedge w.sid=s.sid \wedge a.eid=e.eid \wedge a.day=\text{”mon} \wedge a.startt \leq w.start \wedge a.endt \geq w.end) \}$$

3. Find all cashiers available from 14:00 to 21:00 on all weekdays

Referenced Relations

position(*pid, title, location) ← position(*id, title, location)

has_position(*eid, *pid, acq, rem, prim, train) ← has_position(*employee_id, *position_id, date_acquired, date_removed, is_primary, is_training)

availability(*eid, day, startt, endt) ← availability(*employee_id, day, startt, endt)

employment_history(*eid, emp, unemp) ← employment_history(*employee_id, date_employed, date_unemployed)

$$\{ e \mid employee(e) \wedge (\exists h_p)(\exists p)(\exists e_h)(\exists a_{mon})(\exists a_{tues})(\exists a_{wed})(\exists a_{thurs})(\exists a_{fri})$$
$$(has\_position(h_p) \wedge position(p) \wedge employment\_history(e_h) \wedge p.title = \text{“cashier”} \wedge availability(a_{mon}) \wedge availability(a_{tues}) \wedge availability(a_{wed}) \wedge availability(a_{thurs}) \wedge availability(a_{fri}) \wedge p.pid = h_p.pid \wedge h_p.rem = null \wedge h_p.eid = e.eid \wedge e_h.eid = e.eid \wedge e_h.unemp = null \wedge a_{mon}.day=\text{”mon”} \wedge a_{mon}.eid=e.eid \wedge a_{mon}.startt \leq 14:00 \wedge a_{mon}.endt \geq 14:00 \wedge a_{tues}.day=\text{”tues”} \wedge a_{tues}.eid=e.eid \wedge a_{tues}.startt \leq 14:00 \wedge a_{tues}.endt \geq 14:00 \wedge a_{wed}.day=\text{”wed”} \wedge a_{wed}.eid=e.eid \wedge a_{wed}.startt \leq 14:00 \wedge a_{wed}.endt \geq 14:00 \wedge a_{thurs}.day=\text{”thurs”} \wedge a_{thurs}.eid=e.eid \wedge a_{thurs}.startt \leq 14:00 \wedge a_{thurs}.endt \geq 14:00 \wedge a_{fri}.day=\text{”fri”} \wedge a_{fri}.eid=e.eid \wedge a_{fri}.startt \leq 14:00 \wedge a_{fri}.endt \geq 14:00)$$
$$\}$$

4. Find employees who can work every position in the kitchen, where they are not still training for any of them

Referenced Relations

position(*pid, title, location) ← position(*id, title, location)

has_position(*eid, *pid, acq, rem, prim, train) ← has_position(*employee_id, *position_id, date_acquired, date_removed,is_primary, is_training)

{ e | employee(e) ∧ (∀p)(position(p) ∧ p.location="kitchen"
→ (∃h)(has_position(h) ∧ h.eid=e.eid ∧ h.pid=p.pid)
)
}

5. find all supervisors who can work the earliest supervisor shift on Saturday or Sunday.

Referenced Relations

position(*pid, title, location) ← position(*id, title, location)

shift(*sid, *pid, tname) ← shift(*id,*position_id, task_name)

weekend_hours (sid, start, end) ← weekend_hours(*shift_id, start, end)

has_position(*eid, *pid, acq, rem, prim, train) ← has_position(*employee_id, *position_id, date_acquired, date_removed, is_primary, is_training)

employment_history(*eid, emp, unemp) ← employment_history(*employee_id, date_employed, date_unemployed)

availability(*eid, day, startt, endt) ← availability(*employee_id, day, startt, endt)

{ e | employee(e) ∧
(∃p)(∃hp)(∃a)(∃s1)(∃s2)(∃w1)(∃w2)
(position(p) ∧ has_position(hp) ∧ availability(a) ∧ shift(s1) ∧ shift(s2) ∧
weekend_hours(w1) ∧ weekend_hours(w2) ∧ (a.day="sat" ∨ a.day="sun") ∧
a.eid=e.eid a.startt≤w1.start ∧ a.endt≥w1.end ∧ w1.sid=s1.sid ∧ w2.sid=s2.sid ∧
p.title="supervisor" ∧ s1.pid=p.pid ∧ s2.pid=p.pid ∧ w1.start≤w2.start ∧ hp.eid=e.eid
∧ hp.pid=s1.pid
)
}

6. List the employees who have the second longest request for vacation

Referenced Relations

```
requests_vacation(*rid, *requested, *responded, approved, startD, startT, endD, endT) ←
requests_vacation(*id, *requested_by, *responded_by, is_approved, start_date, start_time,
end_date,end_time)
```

$$
\begin{aligned}
&\{ e \mid employee(e) \wedge (\exists r)(requests\_vacation(r) \wedge r.requested = e.eid \wedge (\exists r2) \\
&\quad (requests\_vacation(r2) \wedge (r2.endD - r2.startD) > (r.endD - r.startD) \wedge \\
&\quad\quad r.rid \neq r2.rid \wedge \\
&\quad\quad \neg (\exists r3)(requests\_vacation(r3) \wedge r3.endD\ r3.startD < r.endD - r.startD \wedge \\
&\quad\quad\quad r3.endD - r3.startD \neq r2.endD\ r2.startD \\
&\quad\quad) \\
&\quad) \\
&) \\
&\}
\end{aligned}
$$

7. List the possible shifts that “John Doe” can work, given his availability and training

Referenced Relations

shift(*sid, *pid, tname) ← shift(*id,*position_id, task_name)

employee(*eid, fname, lname, manager, bday, maxhrswk, phone, clock) ← employee(*id, first_name, last_name, is_manager, birth_date, max_hours_per_week, phone_number, clock_number)

has_position(*eid, *pid, acq, rem, prim, train) ← has_position(*employee_id, *position_id, date_acquired, date_removed, is_primary, is_training)

availability(*eid, day, startt, endt) ← availability(*employee_id, day, startt, endt)

weekday_hours (sid, start, end) ← weekday_hours(*shift_id, wdStart, wdEnd)

weekend_hours (sid, start, end) ← weekend_hours(*shift_id, weStart, weEnd)

$$\{ s \mid shift(s) \wedge (\exists e)(\exists h_p)(\exists a)(\exists w_d)(\exists w_e)(employee(e) \wedge has\_position(h_p) \wedge availability(a) \wedge weekday\_hours(w_d) \wedge weekend\_hours(w_e) \wedge e.fname = \text{“John”} \wedge e.lname = \text{“Doe”} \wedge e.eid = h_p.eid \wedge h_p.train = false \wedge h_p.rem = null \wedge s.pid = h_p.pid \wedge (s.sid = w_d.sid \vee s.sid = w_e.sid) \wedge a.eid=e.eid \wedge w_d.wdStart \geq a.startt \wedge w_d.wdEnd \leq a.endt) \}$$

8. Find all employees who are trained in exactly 2 different positions, who can work with all other employees.

Referenced Relations

```
has_position(*eid, *pid, acq, rem, prim, train) ← has_position(*employee_id,
*position_id, date_acquired, date_removed, is_primary, is_training)

employee(*eid, fname, lname, manager, bday, maxhrswk, phone, clock) ← employee(*id,
first_name, last_name, is_manager, birth_date, max_hours_per_week, phone_number,
clock_number)

cannot_work_with(*e1, *e2) ← cannot_work_with(*employee1_id, *employee2_id)
```

$$
\begin{aligned}
&\{ e \mid employee(e) \wedge (\exists p1)(\exists p2)(has\_position(p1) \wedge has\_position(p2) \wedge p1.pid!=p2.pid \wedge \\
&\qquad p1.eid=e.eid \wedge p2.eid=e.eid \wedge \\
&\qquad \neg (\exists c)\ (cannot\_work\_with(c) \wedge c.e1=e.eid \vee c.e2=e.eid) \\
&\qquad) \\
&\}
\end{aligned}
$$

9. Find sold menu items that used ingredients ordered between 01/01/15 and 01/31/15.

Referenced Relations

```
orders(*eid, *iid, date, quantity) ← orders(*employee_id, *ingredient_id, date, quantity)

used_in(*mid, *iid, quantity) ← used_in(*menu_item_id, *ingredient_id, quantity)

sold_in(*mid, *tid, quantity) ← sold_int(*menu_item_id, *transaction_id, quantity)
```

$$
\begin{aligned}
&\{ m \mid menu\_item(m) \wedge (\forall o)(orders(o) \wedge o.date \geq 01/01/15 \wedge o.date \leq 01/31/1 \\
&\qquad \rightarrow (\exists u)(used\_in(u) \wedge u.iid = o.iid \wedge (Es)(sold\_in(s) \wedge s.mid=u.mid \wedge \\
&\qquad\qquad s.mid=m.mid \\
&\qquad\qquad) \\
&\qquad) \\
&\quad) \\
&\}
\end{aligned}
$$

10. Find all employees who have been scheduled exactly once for a cashier shift

Referenced Relations

position(*pid, title, location) ← position(*id, title, location)

shift(*sid, *pid, tname) ← shift(*id,*position_id, task_name)

is_scheduled_for(*eid, *sid, date) ← is_scheduled_for(*employee_id, *shift_id, date)

employment_history(*eid, emp, unemp) ← employment_history(*employee_id, date_employed, date_unemployed)

$$
\begin{aligned}
\{\ e \mid\ & employee(e) \wedge (\exists p)(\exists s_1)(\exists l_1)(position(p) \wedge shift(s_1) \wedge is\_scheduled\_for(l_1) \wedge p.title=\text{“cashier”} \wedge \\
& \quad s_1.pid=p.pid \wedge l_1.eid=e.eid \wedge l_1.sid=s_1.sid \wedge \\
& \quad \neg(\exists l_2)(\exists s_2)(is\_scheduled\_for(l_2) \wedge shift(s_2) \wedge \\
& \qquad (l_1.sid \neq l_2.sid \vee l_1.date \neq l_2.date) \wedge \\
& \qquad l_2.eid=e.eid \wedge s2.pid=p.pid \wedge l_2.sid=s_2.sid \\
& \quad) \\
&) \\
\}
\end{aligned}
$$

2.4.4 Domain Relational Calculus Expressions for Queries

Domain relational calculus is another form of relational calculus. Domain relational calculus is similar to tuple relational calculus with the exception of what the variable represents. In tuple relational calculus, the variables represent relation tuples, whereas in domain relational calculus, the variables represent domains, or the possible values that an attribute may take within each tuple. The variables iterate through all possible domains quantified for each attribute. In the case where a domain value does not need to be analyzed for the resulting relation, it is represented as a blank space (_), and will not be restricted by any conditions. Like tuple relational calculus, logical connectives may be used, conditions may be nested within conditions, and variables are quantified with the universal or existential quantifiers.

1. Find all employees who have been scheduled to work as a closing janitor.

Referenced Relations
`shift(*id,*position_id, task_name)`
 	
`is_scheduled_for(*employee_id, *shift_id, date)`
 	
`employee(*id, first_name, last_name, is_manager, birth_date, max_hours_per_week, phone_number, clock_number)`
 	
`employment_history(*employee_id, date_employed, date_unemployed)`

{ e | employee(e,_,_,_,_,_,_) ∧ (∃s)
(shift(s,_,"closing janitor") ∧ is_scheduled_for(e,s,_) ∧
`employment_history(e,_,null)`
)
}

2. Find all employees who can work the task named register1 on Mondays.

Referenced Relations

```
position(*id, title, location)

shift(*id,position_id, task_name)

weekday_hours(*shift_id, start, end)

has_position(*employee_id, *position_id, date_acquired, date_removed, is_primary,
is_training)

employment_history(*employee_id, date_employed, date_unemployed)

availability(*employee_id, day, startt, endt)
```

$$\{ e \mid employee(e,\_,\_,\_,\_,\_,\_,\_) \wedge (\exists s_t)(\exists e_t)(\exists s)(\exists p)$$
$$(position(p,\_,\_) \wedge employment\_history(e,\_,null) \wedge has\_position(e,p,\_,null,\_,\_) \wedge$$
$$shift(s,p,\text{"register1"}) \wedge weekday\_hours(s,s_t,e_t) \wedge availability(e,\text{"mon"},<s_t, >e_t,)$$
$$)$$
$$\}$$

3. Find all cashiers available from 14:00 to 21:00 on all weekdays

Referenced Relations

```
position(*id, title, location)

has_position(*employee_id, *position_id, date_acquired, date_removed, is_primary,
is_training)

availability(*employee_id, day, startt, endt)

employment_history(*employee_id, date_employed, date_unemployed)
```

$$\{ e \mid employee(e,\_,\_,\_,\_,\_,\_,\_) \wedge (\exists p)($$
$$position(p,\text{"cashier"},\_) \wedge has\_position(e,p,\_,null,\_,\_) \wedge$$
$$availability(e, \text{“mon”}, \leq 14{:}00, \geq 21{:}00) \wedge$$
$$availability(e, \text{“tues”}, \leq 14{:}00, \geq 21{:}00) \wedge$$
$$availability(e, \text{“wed”}, \leq 14{:}00, \geq 21{:}00) \wedge$$
$$availability(e, \text{“thurs”}, \leq 14{:}00, \geq 21{:}00) \wedge$$
$$availability(e, \text{“fri”}, \leq 14{:}00, \geq 21{:}00) \wedge$$
$$employment\_history(e,\_,null)$$
$$)$$
$$\}$$

4. Find employees who can work every position in the kitchen

Referenced Relations

```
position(*id, title, location)

has_position(*employee_id, *position_id, date_acquired, date_removed, is_primary,
is_training)
```

$\{ e \mid employee(e,\_,\_,\_,\_,\_,\_,\_) \wedge (\forall p)(position(p,\_,\text{“kitchen”}) \rightarrow (\exists h)(has\_position(e,p,\_,\_,\_,\_)) \}$

5. find all supervisors who can work the earliest supervisor shift on Saturday or Sunday.

Referenced Relations

```
position(*id, title, location)

shift(*id,*position_id, task_name)

weekend_hours(*shift_id, start, end)

employee(*id, first_name, last_name, is_manager, birth_date, max_hours_per_week,
phone_number, clock_number)

has_position(*employee_id, *position_id, date_acquired, date_removed, is_primary,
is_training)

employment_history(*employee_id, date_employed, date_unemployed)

availability(*employee_id, day, startt, endt)
```

$\{ e \mid employee(e,\_,\_,\_,\_,\_,\_,\_) \wedge (\exists p)(\exists s1)(\exists s2)(\exists st_1)(\exists st_2)$
$(position(p, \text{“supervisor”},\_) \wedge$
$has\_position(e,p,\_,\_,\_,\_) \wedge$
$availability(e,\text{”sat” } \vee \text{ “sun”}, \leq w1.start, \geq w1.end) \wedge$
$shift(s1, p) \wedge shift(s2, p) \wedge$
$weekend\_hours(s1,st_1,\_) \wedge$
$weekend\_hours(s2,st_2,\_) \wedge st_1 < st_2 \wedge$
$has\_position(e,p,\_,\_,\_,\_)$
$)$
$\}$

6. List the employees who have the second longest request for vacation

Referenced Relations

```
requests_vacation(*id, *requested_by, *responded_by, is_approved, start_date, start_time,
end_date, end_time)
```

$$
\begin{aligned}
&\{ e \mid employee(e,\_,\_,\_,\_,\_,\_,\_) \wedge (\exists r1)(\exists r2)(\exists s1_s)(\exists s1_e)(\exists s2_s)(\exists s2_e)(\exists s3_s)(\exists s3_e) \\
&\quad (requests\_vacation(r1, e, \_, s1_s, s1_e, \_, \_) \wedge \\
&\quad requests\_vacation(r2, e, \_, s2_s, s2_e, \_, \_) \wedge \\
&\quad s2_e - s2_s < s1s_s - s2_e \wedge r! = r2 \wedge \\
&\quad s2_e - s2_s < s1s_s - s2_e \wedge \\
&\quad \neg (\exists r3)(requests\_vacation(r3, e, s3_s, s3_e, \_, \_) \wedge \\
&\qquad s3_e - s3_s < s1_e - s1_s \wedge s3_e - s3_s != s2_e - s2_s \\
&\qquad) \\
&\quad) \\
&\}
\end{aligned}
$$

7. List the possible shifts that “John Doe” can work, given his availability and training

Referenced Relations

```
shift(*id,*position_id, task_name)

employee(*id, first_name, last_name, is_manager, birth_date, max_hours_per_week,
phone_number, clock_number)

has_position(*employee_id, *position_id, date_acquired, date_removed, is_primary,
is_training)

availability(*employee_id, day, startt, endt)

weekday_hours(*shift_id, wdStart, wdEnd)

weekend_hours(*shift_id, weStart, weEnd)
```

$$
\begin{aligned}
&\{ s \mid shift(s,\_,\_) \wedge (\exists e)(\exists s_{wd})(\exists e_{wd})(\exists s_{we})(\exists e_{we}) \\
&\qquad (employee(e,\text{“John”},\text{“Doe”},\_,\_,\_,\_,\_) \wedge \\
&\qquad has\_position(e,p,\_,null,\_,false) \wedge \\
&\qquad (weekday\_hours(s, s_{wd}, e_{wd}) \vee weekend\_hours(s, s_{we}, e_{we})) \\
&\qquad \wedge availability(e,\_,(\leq s_{wd} \vee \leq s_{wd}),(\geq e_{wd} \vee \geq e_{wd})) \\
&\qquad) \\
&\}
\end{aligned}
$$

8. Find all employees who are trained in exactly 2 different positions, who can work with all other employees - these are more valuable employees

Referenced Relations

```
has_position(*employee_id, *position_id, date_acquired, date_removed, is_primary,
is_training)

employee(*id, first_name, last_name, is_manager, birth_date, max_hours_per_week,
phone_number, clock_number)

cannot_work_with(*employee1_id, *employee2_id)
```

$$\{ e \mid employee(e,\_,\_,\_,\_,\_,\_,\_) \wedge (\exists p1)(\exists p2)$$
$$(has\_position(e,p1,\_,\_,\_,\_) \wedge has\_position(e,p2,\_,\_,\_,\_) \wedge p1 \neq p2 \wedge$$
$$\neg (cannot\_work\_with(e,\neq e))$$
$$)$$
$$\}$$

9. Find sold menu items that used ingredients ordered between 01/01/15 and 01/31/15.

Referenced Relations

```
orders(*employee_id, *ingredient_id, date, quantity)

used_in(*menu_item_id, *ingredient_id, quantity)

sold_in(*menu_item_id, *transaction_id, quantity)
```

$$\{ m \mid menu\_item(m,\_,\_,\_,\_) \wedge (\forall d)(\exists i)($$
$$orders(\_,i,d,\_) \wedge d \geq 01/01/15 \wedge d \leq 01/31/15 \rightarrow$$
$$used\_in(m,i,\_) \wedge sold\_in(m,\_,\_)$$
$$)$$
$$\}$$

10. Find all employees who have been scheduled exactly once for a cashier shift

Referenced Relations

```
position(*id, title, location)

shift(*id,*position_id, task_name)

employee(*id, first_name, last_name, is_manager, birth_date, max_hours_per_week,
phone_number, clock_number)

is_scheduled_for(*employee_id, *shift_id, date)

employment_history(*employee_id, date_employed, date_unemployed)
```

$$
\begin{aligned}
&\{ e \mid employee(e,\_,\_,\_,\_,\_,\_,\_) \wedge (\exists p)(\exists s_1)(\exists d_1)(\exists d_2) \\
&\quad (position(p,\text{“cashier”},\_) \wedge shift(s_1,p,\_) \wedge is\_scheduled\_for(e,s_1,d_1) \wedge \\
&\quad \neg(\exists s_2)(is\_scheduled\_for(e,s_2,d_2) \wedge shift(s_2,p,\_) \wedge d_1 != d_2 \vee s_1 != s_2) \\
&\quad) \\
&\}
\end{aligned}
$$

Phase 3 Oracle Database Mangement System

Section 3.1 Normalization of Relations

This section describes some design techniques for databases. It lists possible problems, or update anomalies, and ways to avoid them, called normalization. Several forms of normalization are described.

3.1.1 Documentation for Normalization and Normal Forms

A good database design is logically easy for users to follow, and implemented with consideration for the physical storage of the data. A quality relational database schema can be designed by following four general guidelines. The first is that attribute names should communicate their meaning as clearly as possible. The second is that redundant information is reduced. Redundancy reduction saves precious storage space. The third is that null values are reduced as much as possible. Attributes that frequently hold null values are problematic because they store extra space with less meaning, and are unpredictable when used in aggregate functions and joins. Finally, the fourth is that excess tuples that contain much of the same information (spurious tuples) are not created.

A well designed database avoids update anomalies. Update anomalies complicate data operations and result in unnecessary complicated insertion, accidental data loss, or accidental data inconsistencies. These three conditions are called Insertion Anomalies, Deletion Anomalies, and Modification Anomalies. Insertion Anomalies are when inserting new data in a relation requires extra data in the same tuple to avoid null attributes. Deletion Anomalies are when deleting certain data leads to the unintentional deletion of additional data. Modification Anomalies are when changing the value in one relation requires updating other relations so that relations will all remain consistent.

Normalization of a database aims to minimize data redundancy and avoid update anomalies. Normalization relies on the concept of functional dependency. A functional dependency is a constraint between two sets of attributes from the database, which is determined by the database designer who knows the semantics of all the attributes. A functional dependency is disproved by a single counterexample. For example, if there are two subsets of attributes X and Y in a relation, and

X -> Y holds true for every tuple, it is considered a functional dependency. There are four main categories of normal forms: First normal form (1NF), Second normal form (2NF), Third normal form (3NF), and Boyce-Codd normal form (BNF). Each subsequent form contains all of the characteristics of its predecessors. The classification of a relation as a normal form refers to the highest normal form that it meets.

First normal form is simply the basic flat relational model where there are no composite or multi-valued attributes. Second normal form is based on full functional dependency. Each relation has both a subset of prime attributes {p} (ones that make up the primary key), and non prime attributes {n} (all other attributes). The subset of nonprime attributes must be functionally dependent upon the prime attributes. So it is not the case where attributes can be removed from {p} and {n} still holds as functionally dependent of {p}. Third normal form is based on transitive dependencies. Transitive dependencies are when there are three subsets of attributes in a relation {a}, {b}, and {c} where {a} is the primary key. If {c} is dependent on {b} and {b} is dependent on {a}, then this is a transitive dependency where {c} is dependent on {a} through {b}. There can be no transitive dependencies in 3NF. Boyce-Codd normal form is a stricter form of 3NF. If there is a non-trivial transitive dependency, it must be pulled out of the relation in order for it to qualify for Boyce-Codd normal form.

Example of relation not in 1NF (nested relations)

```
Employee(id, {first_name, last_name}, is_manager, birth_date, max_hours_per_week,
phone_number, clock_number)
```

Possible Anomalies

There are technically no anomalies that may occur, because a relation that is not in 1NF does not even qualify as a relation, and queries to insert, update or delete data are not valid.

Conversion to 1NF

This relation may be “flattened” so that the attributes are pulled out into the parent relation. This is a good option in this situation, because the number of attributes is known, and they are not likely to hold null values.

Resulting 1NF Relation

```
Employee(id, first_name, last_name, is_manager, birth_date, max_hours_per_week,
phone_number, clock_number)
```

Example of relation not in 2NF

```
Ingredient_Supplier(*ingredient_name, *supplier_name, delivery_days)
```

Possible Anomalies

This relation stores each ingredient's name, and the supplier it comes from. Insertion and modification anomalies are possible with this relation, both of which result from the functional dependency of delivery_days on supplier. The primary key is {ingredient_name, supplier}. Insertion anomalies result from having to re-enter the same delivery_days values for suppliers that deliver more than one ingredient. Modification anomalies result from having to change multiple delivery_days values when a supplier changes their delivery schedule, due to that value's functional dependency upon supplier.

Conversion to 2NF

The relation is split into two tables. One table associates suppliers with the ingredients they deliver, the other associates suppliers with the days they deliver their ingredients.

Resulting 2NF Relation

```
Ingredient(*ingredient_name, *supplier_id)
Supplier(*supplier_id, supplier_name, delivery_days)
```

Example of relation not in 3NF

```
Empl_Dept(*id, ssn, f_name, l_name, phone, address, dept_no, dept_location)
```

Possible Anomalies

This relation stores each employee's information, and the department the employee works for, with the employee's id as the primary key. Insertion, deletion, and modification anomalies are possible with this relation, all of which result from the functional dependency of dept_location on id through dept_no. Insertion anomalies result from having to re-enter the same delivery_location values for employees that work for the same department. Deletion anomalies result from losing information about a department when there are no employees working for that department. Modification anomalies result from having to change multiple dept_location values when a department changes location, or when a department changes a group of employees.

Conversion to 2NF

The relation is split into two tables. One table stores employee information, and the other stores department information, with the department and employee works tracked through dept_no only in the employee table.

Resulting 2NF Relation

```
Employee(*id, ssn, f_name, l_name, phone, address, dept_no)
Supplier(*dept_no, dept_location)
```

3.1.2 Normal Forms for this database

This section shows all relations that were already in Boyce-Codd Normal Form or 3rd Normal Form. It also shows how relations not already in these two forms were converted into them.

Relation	Form
availability (employee_id, day, startt, endt)	BCNF
cannot_work_with (employee1_id, employee2_id)	BCNF
delivered_by (supplier, ingredient_id, date_delivered, quantity)	BCNF
employee (id, first_name, last_name, is_manager, birth_date, max_hours_per_week, phone_number, clock_number)	BCNF
employment_history (employee_id, date_employed, date_unemployed)	BCNF
has_position (employee_id, position_id, date_acquired, date_removed, is_primary, is_training)	BCNF
holiday (date, name, type)	3NF
ingredient (id, name)	BCNF
is_scheduled_for (employee_id, shift_id, date)	BCNF
menu_item (id, name, type, price, photo)	3NF
orders (employee_id, ingredient_id, date, quantity)	BCNF
position (id, title, location)	BCNF
requests_vacation (requested_by, responded_by, is_approved, start_date_time, end_date_time)	BCNF
shift (id, position_id, task_name)	BCNF
sold_in (menu_item_id, transaction_id, quantity)	BCNF
supplier (id, title, delivery_days)	BCNF
transaction (id, date)	BCNF
used_in (menu_item_id, ingredient_id, quantity)	BCNF
user (employee_id, email, password, api_token)	BCNF
weekday_hours (shift_id, start_time, end_time)	BCNF
weekend_hours (shift_id, start_time, end_time)	BCNF

Section 3.2 SQL*PLUS: Main Purpose and Functionality Explained

SQL is a language designed for managing data in a Relational DBMS. The language is based on relational algebra and tuple relational calculus, and is used to handle data definitions, data manipulation, and data control. Data definitions consist of statements that allow for the creation, deletion, and updating of relation definitions. These statements essentially define tables and their attributes, which can then be filled with tuples, or rows. Data manipulation statements are those commands that allow table rows to be inserted, updated, removed, or queried. These are the instructions normally executed by a front-end user program that facilitates interaction between the database and its users. Finally, data control statements are those commands that alter access to the database. These commands use 'grant' and 'revoke' statements to alter which users have direct access to the database, and the commands they can perform. SQL/PLUS, the implementation of SQL normally included by Oracle DBMS, extends SQL by allowing for batch queries and an interactive command line interface. Batch queries are simply groupings of one or more SQL statements, which improves the effectiveness sending instructions to the database and receiving results from it, especially when executed over a computer network. (Oracle, SQL*PLUS Quick Start, 2015) (Microsoft, 2015) (SQL, Wikipedia, 2015)

Section 3.3 Schema Objects for Oracle DBMS

Schemas in Oracle are collections of logical data structures called "schema objects". These objects can have many different functions, but generally work to compartmentalize the functionality of a database into different classes, which can then be instantiated into objects and put to specific use. Each user owns one schema, named as their own user name, and each schema can contain many schema objects. These objects are logically and physically contained within their associated data tablespace, which is a simply a subgrouping of schema objects within a database schema. The following schema objects may be created: (Oracle, Schema Objects, 2015)

- Tables
- Views
- Materialized Views
- Dimensions
- Sequence Generators
- Synonyms
- Indexes
- Index-Organized Tables
- Application Domain Indexes
- Clusters

- Hash Clusters

Tables

Tables are used as the basic unit of storage for a database. They contain rows which are analogous to facts in the database miniworld that have the values for the attributes, or column fields, in the table. The basic information needed to create a table is the table name, column names, column data types, and potential domain restrictions for a column's data type. Data (rows) can be inserted into tables, deleted from them, updated, or queried to be viewed.

Views

Views are meant to act as a virtual table, which can be interpreted as the stored result of a query. This object essentially takes the result of a query and treats it as a table, thereby acting to provide a tailored presentation of one or more tables. For the most part, data in views can also be modified via create, delete, update, and view commands, which act by modifying data in some base table. However, logical constraints, such as triggers, are not supported for views.

Materialized Views

These objects are used to store aggregate or replicate data regarding one or more tables. Materialized views essentially store the results of an aggregate query. These objects provide the primary benefit of making aggregate data readily available to users, and also speed up the performance of query rewrites. The Oracle optimizer may create some materialized views automatically to speed up database queries, since they contain some time-saving functionality such as the ability to perform joins with needing to aggregate.

Dimensions

The purpose of dimensions is to create a hierarchy relationship between either a pair of columns or a set of columns. This creates a relationship that associates the child column in the hierarchy with exactly one value in its parent column. In the case columns come from the same table, the dimension is considered denormalized. If the columns come from multiple tables, the dimension is either partially or fully normalized.

Sequence Generators

This object provides a sequential series of numbers to be inserted into table values. Note, it is not meant to be used to store a sequence of numbers itself. It is meant to be used in a multiuser environment when multiple users are attempting to insert data with unique attribute values that are sequential. Its purpose is to speed up data entry by automatically generating and providing the correct key values to each user, so they do not have to wait for one another to determine which primary key values they should use next. Defining a sequence generator requires the sequence name, whether the sequence ascends or descends, the interval between numbers, and whether or not sets of numbers should be cached.

Synonyms

Synonyms are aliases for other schema objects, procedures, functions, packages, types, user-defined objects, or other synonyms. They are used for security and convenience, since they can be used to mask an object's name and owner, help transparently define the location of an object in a distributed database, simplify SQL statements, or enable restricted access to similar but specialized objects. Synonyms can be made both private or public.

Indexes

An index provides a faster access path to specified table columns, while remaining logically and physically independent from the table. They are used to speed up queries on the table in question. When used correctly, indexes are the primary means of improving disk I/O speeds. A column may appear in more than one index, as long as each index contains a unique set of columns. The following indexing schemes are provided, with corresponding speed improvements:

- B-tree indexes
- B-tree cluster indexes
- Hash cluster indexes
- Reverse key indexes
- Bitmap indexes
- Bitmap join indexes
- Note, the presence of many indexes on a table decrease performance of updates, insertions, and deletions on that table, since many more object updates are required.

Index-Organized Tables

An index-organized table is essentially a B-tree index that represents a table, because all the table information is stored within the index, thus saving storage space and shortening look-up times for the entire table. It is useful for tables where the only attributes are the primary key and the non-key attributes that are functionally dependent upon the key, and the only index needed is on all those attributes taken together. The object is generally created for tables in 3rd Normal Form that contain

exactly one functionally dependent non-key attribute. In practice, such tables are those that end up being used as reference data, such as code look-ups. (APC, 2015)

Application Domain Indexes

Certain data types cannot effectively be stored with the default index types provided by Oracle. For these complex data types, an Application Domain Index should be defined instead. The object allows for customized indexes whose data structure is pre-defined in some application software, which is denoted the “cartridge”. This allows for extensible indexing, or creating “indextypes”, to encapsulate application-specific index management routines.

Clusters

For the cases where multiple tables share one or more attributes, and are used together often, the cluster schema object is available to more effectively stored table data and speed up joins. The object works by creating “cluster key values” from the shared attribute, which a set of the values that appear in all instance of the attribute. These key values are then associated with their corresponding rows in each, thereby compressing the data and storing the linking metadata necessary to perform a join.

Hash Clusters

It is also possible to create a cluster that is stored by the result of a hash function on the cluster key values mentioned above. This is ideal for tables that are often queried for a selection on an equality (e.g. hNo = 5). The object works by first creating the hash cluster, and then loading tables into it. Note, at least two I/O operations must be performed for any tables stored in the hash cluster, one to locate the rows in question, and another to perform the read/write.

Section 3.4 List Relations With SQL Commands

This section describes each relation using the SQL commands DESC <table> and SELECT * FROM <table>. The results of these commands for each table are listed below.

```
CS342 SQL> DESC BAJS_AVAILABILITY;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- ------------------------------------------------
 EMPLOYEE_ID                                                             NOT NULL NUMBER(5)
 DAY                                                                     NOT NULL VARCHAR2(8)
 STARTT                                                                  NOT NULL NUMBER(4)
 ENDT                                                                             NUMBER(4)

CS342 SQL> SELECT * FROM BAJS_AVAILABILITY;

EMPLOYEE_ID DAY         STARTT       ENDT
----------- -------- ---------- ----------
          1 mon            600       2100
          2 mon            600       2100
          3 mon            600       2100
          4 mon            600       2100
          5 mon            600       2100
          6 mon            600       2100
          7 mon            600       2100
          8 mon            600       2100
          9 mon            600       2100
         10 mon            600       2100
         11 mon            600       2100
         12 mon            600       2100
         13 mon            600       2100
         14 mon            600       2100
         15 mon            600       2100
         16 mon            600       2100
         17 mon            600       2100
         18 mon            600       2100
         19 mon            600       2100
         20 mon            600       2100
         21 mon            600       2100
         22 mon            600       2100
         23 mon            600       2100
         24 mon            600       2100
         25 mon            600       2100
         26 mon            600       2100
         27 mon            600       2100
         28 mon            600       2100
         29 mon            600       2100
         30 mon            600       2100
         31 mon            600       2100
         32 mon            600       2100
         33 mon            600       2100
         34 mon            600       2100
         35 mon            600       2100
         36 mon            600       2100
         37 mon            600       2100

EMPLOYEE_ID DAY         STARTT       ENDT
----------- -------- ---------- ----------
         38 mon            600       2100
         39 mon            600       2100
          1 tue            600       2100
          2 tue            600       2100
          3 tue            600       2100
          4 tue            600       2100
          5 tue            600       2100
          6 tue            600       2100
          7 tue            600       2100
          8 tue            600       2100
```

```
         9 tue            600       2100
        10 tue            600       2100
        11 tue            600       2100
        12 tue            600       2100
        13 tue            600       2100
        14 tue            600       2100
        15 tue            600       2100
        16 tue            600       2100
        17 tue            600       2100
        18 tue            600       2100
        19 tue            600       2100
        20 tue            600       2100
        21 tue            600       2100
        22 tue            600       2100
        23 tue            600       2100
        24 tue            600       2100
        25 tue            600       2100
        26 tue            600       2100
        27 tue            600       2100
        28 tue            600       2100
        29 tue            600       2100
        30 tue            600       2100
        31 tue            600       2100
        32 tue            600       2100
        33 tue            600       2100
        34 tue            600       2100
        35 tue            600       2100

EMPLOYEE_ID DAY         STARTT       ENDT
----------- -------- ---------- ----------
        36 tue            600       2100
        37 tue            600       2100
        38 tue            600       2100
        39 tue            600       2100
         1 wed            600       2100
         2 wed            600       2100
         3 wed            600       2100
         4 wed            600       2100
         5 wed            600       2100
         6 wed            600       2100
         7 wed            600       2100
         8 wed            600       2100
         9 wed            600       2100
        10 wed            600       2100
        11 wed            600       2100
        12 wed            600       2100
        13 wed            600       2100
        14 wed            600       2100
        15 wed            600       2100
        16 wed            600       2100
        17 wed            600       2100
        18 wed            600       2100
        19 wed            600       2100
        20 wed            600       2100
        21 wed            600       2100
        22 wed            600       2100
        23 wed            600       2100
        24 wed            600       2100
        25 wed            600       2100
        26 wed            600       2100
        27 wed            600       2100
        28 wed            600       2100
        29 wed            600       2100
        30 wed            600       2100
        31 wed            600       2100
        32 wed            600       2100
        33 wed            600       2100

EMPLOYEE_ID DAY         STARTT       ENDT
----------- -------- ---------- ----------
        34 wed            600       2100
        35 wed            600       2100
```

```
          36 wed            600       2100
          37 wed            600       2100
          38 wed            600       2100
          39 wed            600       2100
           1 thu            600       2100
           2 thu            600       2100
           3 thu            600       2100
           4 thu            600       2100
           5 thu            600       2100
           6 thu            600       2100
           7 thu            600       2100
           8 thu            600       2100
           9 thu            600       2100
          10 thu            600       2100
          11 thu            600       2100
          12 thu            600       2100
          13 thu            600       2100
          14 thu            600       2100
          15 thu            600       2100
          16 thu            600       2100
          17 thu            600       2100
          18 thu            600       2100
          19 thu            600       2100
          20 thu            600       2100
          21 thu            600       2100
          22 thu            600       2100
          23 thu            600       2100
          24 thu            600       2100
          25 thu            600       2100
          26 thu            600       2100
          27 thu            600       2100
          28 thu            600       2100
          29 thu            600       2100
          30 thu            600       2100
          31 thu            600       2100

EMPLOYEE_ID DAY        STARTT      ENDT
----------- -------- ---------- ----------
          32 thu            600       2100
          33 thu            600       2100
          34 thu            600       2100
          35 thu            600       2100
          36 thu            600       2100
          37 thu            600       2100
          38 thu            600       2100
          39 thu            600       2100
           1 fri            600       2100
           2 fri            600       2100
           3 fri            600       2100
           4 fri            600       2100
           5 fri            600       2100
           6 fri            600       2100
           7 fri            600       2100
           8 fri            600       2100
           9 fri            600       2100
          10 fri            600       2100
          11 fri            600       2100
          12 fri            600       2100
          13 fri            600       2100
          14 fri            600       2100
          15 fri            600       2100
          16 fri            600       2100
          17 fri            600       2100
          18 fri            600       2100
          19 fri            600       2100
          20 fri            600       2100
          21 fri            600       2100
          22 fri            600       2100
          23 fri            600       2100
          24 fri            600       2100
          25 fri            600       2100
          26 fri            600       2100
```

```
        27 fri            600       2100
        28 fri            600       2100
        29 fri            600       2100

EMPLOYEE_ID DAY        STARTT       ENDT
----------- -------- ---------- ----------
        30 fri            600       2100
        31 fri            600       2100
        32 fri            600       2100
        33 fri            600       2100
        34 fri            600       2100
        35 fri            600       2100
        36 fri            600       2100
        37 fri            600       2100
        38 fri            600       2100
        39 fri            600       2100
         1 sat            600       2100
         2 sat            600       2100
         3 sat            600       2100
         4 sat            600       2100
         5 sat            600       2100
         6 sat            600       2100
         7 sat            600       2100
         8 sat            600       2100
         9 sat            600       2100
        10 sat            600       2100
        11 sat            600       2100
        12 sat            600       2100
        13 sat            600       2100
        14 sat            600       2100
        15 sat            600       2100
        16 sat            600       2100
        17 sat            600       2100
        18 sat            600       2100
        19 sat            600       2100
        20 sat            600       2100
        21 sat            600       2100
        22 sat            600       2100
        23 sat            600       2100
        24 sat            600       2100
        25 sat            600       2100
        26 sat            600       2100
        27 sat            600       2100

EMPLOYEE_ID DAY        STARTT       ENDT
----------- -------- ---------- ----------
        28 sat            600       2100
        29 sat            600       2100
        30 sat            600       2100
        31 sat            600       2100
        32 sat            600       2100
        33 sat            600       2100
        34 sat            600       2100
        35 sat            600       2100
        36 sat            600       2100
        37 sat            600       2100
        38 sat            600       2100
        39 sat            600       2100
         1 sun            600       2100
         2 sun            600       2100
         3 sun            600       2100
         4 sun            600       2100
         5 sun            600       2100
         6 sun            600       2100
         7 sun            600       2100
         8 sun            600       2100
         9 sun            600       2100
        10 sun            600       2100
        11 sun            600       2100
        12 sun            600       2100
        13 sun            600       2100
        14 sun            600       2100
```

```
        15 sun            600       2100
        16 sun            600       2100
        17 sun            600       2100
        18 sun            600       2100
        19 sun            600       2100
        20 sun            600       2100
        21 sun            600       2100
        22 sun            600       2100
        23 sun            600       2100
        24 sun            600       2100
        25 sun            600       2100

EMPLOYEE_ID DAY         STARTT       ENDT
----------- -------- ---------- ----------
        26 sun            600       2100
        27 sun            600       2100
        28 sun            600       2100
        29 sun            600       2100
        30 sun            600       2100
        31 sun            600       2100
        32 sun            600       2100
        33 sun            600       2100
        34 sun            600       2100
        35 sun            600       2100
        36 sun            600       2100
        37 sun            600       2100
        38 sun            600       2100
        39 sun            600       2100


CS342 SQL> DESC BAJS_AVAILABILITY;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- --------
 EMPLOYEE_ID                                                             NOT NULL NUMBER(5)
 DAY                                                                     NOT NULL VARCHAR2(8)
 STARTT                                                                  NOT NULL NUMBER(4)
 ENDT                                                                             NUMBER(4)


CS342 SQL> DESC BAJS_CANNOT_WORK_WITH;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- --------
 EMPLOYEE1_ID                                                                     NUMBER(5)
 EMPLOYEE2_ID                                                                     NUMBER(5)


CS342 SQL> SELECT * FROM BAJS_CANNOT_WORK_WITH;

EMPLOYEE1_ID EMPLOYEE2_ID
------------ ------------
           1            2
           1            3
           1            4
          10           11
          10           12
          15           16
          15           17

7 rows selected.
```

```
CS342 SQL> DESC BAJS_DELIVERED_BY;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- ------------
 SUPPLIER_ID                                                             NOT NULL NUMBER(5)
 INGREDIENT_ID                                                           NOT NULL NUMBER(5)
 DATE_DELIVERED                                                                   DATE
 QUANTITY                                                                         NUMBER(3)

CS342 SQL> SELECT * FROM BAJS_DELIVERED_BY;

SUPPLIER_ID INGREDIENT_ID DATE_DELI   QUANTITY
----------- ------------- --------- ----------
          1             1 04-DEC-02         10
          2             2 05-DEC-02         10
          3             3 06-DEC-02         10
          4             4 07-DEC-02         10
          5             5 08-DEC-02         10
          6             6 09-DEC-02         10
          7             7 10-DEC-02         10
          8             8 11-DEC-02         10
          9             9 12-DEC-02         10
         10            10 13-DEC-02         10

10 rows selected.
CS342 SQL> DESC BAJS_EMPLOYEE;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- ------------
 ID                                                                      NOT NULL NUMBER(5)
 FIRST_NAME                                                                       VARCHAR2(10)
 LAST_NAME                                                                        VARCHAR2(10)
 IS_MANAGER                                                                       NUMBER(1)
 BIRTH_DATE                                                                       DATE
 MAX_HRS_WEEK                                                                     NUMBER(2)
 PHONE_NUMBER                                                                     VARCHAR2(10)
 CLOCK_NUMBER                                                                     NUMBER(2)

CS342 SQL> SELECT * FROM BAJS_EMPLOYEE;

        ID FIRST_NAME LAST_NAME  IS_MANAGER BIRTH_DAT MAX_HRS_WEEK PHONE_NUMB CLOCK_NUMBER
---------- ---------- ---------- ---------- --------- ------------ ---------- ------------
         1 John       Doe               1 04-DEC-99           40 1234567890           1
         2 Jane       Akachi            1 04-DEC-98           40 1234567890           2
         3 Bob        Ping              0 04-DEC-97           40 1234567890           3
         4 Susan      Jordan            0 04-DEC-96           40 1234567890           4
         5 Chris      Tumelo            0 04-DEC-95           40 1234567890           5
         6 Kamala     Agrippa           0 04-DEC-94           40 1234567890           6
         7 Lei        Lindsey           0 04-DEC-93           40 1234567890           7
         8 Sal        Finley            0 04-DEC-92           40 1234567890           8
         9 Taonga     Noam              0 04-DEC-91           40 1234567890           9
        10 Aston      Francis           0 04-DEC-90           40 1234567890          10
        11 Mavuto     Leigh             0 04-DEC-89           40 1234567890          11
        12 Steph      Vivian            0 04-DEC-88           40 1234567890          12
        13 Maria      Shiori            0 04-DEC-87           40 1234567890          13
        14 Deniz      Romilda           0 04-DEC-86           40 1234567890          14
        15 Jules      Riley             0 04-DEC-85           40 1234567890          15
        16 Li         Yuki              0 04-DEC-84           40 1234567890          16
        17 Sammie     Maayan            0 04-DEC-83           40 1234567890          17
        18 Hadley     Khurshid          0 04-DEC-82           40 1234567890          18
        19 Chike      Rayan             0 04-DEC-81           40 1234567890          19
        20 Udo        Hilary            0 04-DEC-80           40 1234567890          20
        21 Udo        Hilary            0 04-DEC-80           40 1234567890          21
        22 Udo        Hilary            0 04-DEC-80           40 1234567890          22
        23 Lenny      Teo               0 04-DEC-80           40 1234567890          23
        24 Zephyrus   Titu              0 04-DEC-80           40 1234567890          24
        25 Shyama     Lennox            0 04-DEC-80           40 1234567890          25
        26 Ulyses     Stiofan           0 04-DEC-80           40 1234567890          26
        27 Paora      Drusus            0 04-DEC-80           40 1234567890          27
        28 Ade        Anar              0 04-DEC-80           40 1234567890          28
        29 Egill      Aldin             0 04-DEC-80           40 1234567890          29
        30 Marcelli   Jerald            0 04-DEC-80           40 1234567890          30
        31 Kasey      Kastor            0 04-DEC-80           40 1234567890          31
        32 Gina       Foster            0 04-DEC-80           40 1234567890          32
```

```
        33 Aminta     Owain             0 04-DEC-80          40 1234567890          33
        34 Devraj     Zaki              0 04-DEC-80          40 1234567890          34
        35 Nikolas    Mudiwa            0 04-DEC-80          40 1234567890          35
        36 Gerhard    Flurry            0 04-DEC-80          40 1234567890          36
        37 Vinzent    Ealaed            0 04-DEC-80          40 1234567890          37

        ID FIRST_NAME LAST_NAME  IS_MANAGER BIRTH_DAT MAX_HRS_WEEK PHONE_NUMB CLOCK_NUMBER
---------- ---------- ---------- ---------- --------- ------------ ---------- ------------
        38 Earl       Hearl             0 04-DEC-80          40 1234567890          38
        39 Ash        Sash              0 04-DEC-80          40 1234567890          39
        40 NEAyden    Vijlem            0 04-DEC-80          40 1234567890          40
        41 NEAlpin    Edward            0 04-DEC-80          40 1234567890          41
        42 NELevi     Mattaus           0 04-DEC-80          40 1234567890          42
        43 NEVolya    Kunibet           0 04-DEC-80          40 1234567890          43
        44 NEBoran    Thijs             0 04-DEC-80          40 1234567890          44
        45 NETheodor  Meade             0 04-DEC-80          40 1234567890          45
        46 NEIndra    Lugel             0 04-DEC-80          40 1234567890          46
        47 NELoic     Olav              0 04-DEC-80          40 1234567890          47
        48 NERoel     Yuki              0 04-DEC-80          40 1234567890          48
        49 NECamden   Ghassen           0 04-DEC-80          40 1234567890          49
        50 NERob      Eemen             0 04-DEC-80          40 1234567890          50

50 rows selected.

CS342 SQL> DESC BAJS_EMPLOYMENT_HISTORY;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- -------
-----------------------------------------
 EMPLOYEE_ID                                                             NOT NULL NUMBER(5)
 DATE_EMPLOYED                                                           NOT NULL DATE
 DATE_UNEMPLOYED                                                                  DATE

CS342 SQL> SELECT * FROM BAJS_EMPLOYMENT_HISTORY;

EMPLOYEE_ID DATE_EMPL DATE_UNEM
----------- --------- ---------
          1 24-DEC-09
          2 24-DEC-11
          3 24-DEC-09
          4 24-DEC-11
          5 24-DEC-90
          6 24-DEC-09
          7 24-DEC-09
          8 24-DEC-10
          9 24-DEC-10
         10 24-DEC-10
         11 24-DEC-09
         12 24-DEC-09
         13 24-DEC-09
         14 24-DEC-08
         15 24-DEC-08
         16 24-DEC-08
         17 24-DEC-08
         18 24-DEC-10
         19 24-DEC-10
         20 24-DEC-10
         21 24-DEC-10
         22 24-DEC-10
         23 24-DEC-10
         24 24-DEC-09
         25 24-DEC-09
         26 24-DEC-89
         27 24-DEC-89
         28 24-DEC-89
         29 24-DEC-89
         30 24-DEC-89
         31 24-DEC-89
         32 24-DEC-89
         33 24-DEC-89
         34 24-DEC-89
         35 24-DEC-89
         36 24-DEC-89
```

```
        37 24-DEC-89

EMPLOYEE_ID DATE_EMPL DATE_UNEM
----------- --------- ---------
         38 24-DEC-89
         39 24-DEC-89
         40 24-DEC-89 24-DEC-99
         41 24-DEC-89 24-DEC-99
         42 24-DEC-89 24-DEC-99
         43 24-DEC-89 24-DEC-99
         44 24-DEC-89 24-DEC-99
         45 24-DEC-89 24-DEC-99
         46 24-DEC-89 24-DEC-00
         47 24-DEC-89 24-DEC-00
         48 24-DEC-89 24-DEC-00
         49 24-DEC-89 24-DEC-00
         50 24-DEC-89 24-DEC-00
          1 24-DEC-08 01-DEC-09
          2 24-DEC-10 01-DEC-11
          3 24-DEC-08 01-DEC-09
          4 24-DEC-10 01-DEC-11
          5 24-DEC-89 01-DEC-99
          6 24-DEC-08 01-DEC-09
          7 24-DEC-08 01-DEC-09
          8 24-DEC-07 01-DEC-08
          9 24-DEC-07 01-DEC-08
         10 24-DEC-07 01-DEC-08

60 rows selected.
CS342 SQL> DESC BAJS_HAS_POSITION;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- -------
-----------------------------------------
 EMPLOYEE_ID                                                             NOT NULL NUMBER(5)
 POSITION_ID                                                             NOT NULL NUMBER(5)
 DATE_ACQUIRED                                                           NOT NULL DATE
 DATE_REMOVED                                                                     DATE
 IS_PRIMARY                                                                       NUMBER(1)
 IS_TRAINING                                                                      NUMBER(1)

CS342 SQL> SELECT * FROM BAJS_HAS_POSITION;

EMPLOYEE_ID POSITION_ID DATE_ACQU DATE_REMO IS_PRIMARY IS_TRAINING
----------- ----------- --------- --------- ---------- -----------
          1           1 01-DEC-11                   1           0
          2           1 01-DEC-11                   1           0
          3           1 01-DEC-11                   1           1
          4           1 02-DEC-11                   1           0
          5           1 02-DEC-11                   1           0
          6           4 02-DEC-11                   1           1
          7           4 03-DEC-11                   1           0
          8           3 03-DEC-11                   1           0
          9           3 03-DEC-11                   1           1
         10           3 04-DEC-11                   1           0
         11           3 04-DEC-11                   1           0
         12           3 04-DEC-11                   1           0
         13           2 03-DEC-11                   1           0
         14           2 03-DEC-11                   1           1
         15           2 04-DEC-11                   1           0
         16           2 04-DEC-11                   1           0
         17           2 04-DEC-11                   1           0
         18           2 04-DEC-11                   1           0
         19           2 04-DEC-11                   1           0
         20           2 04-DEC-11                   1           0
         21           2 04-DEC-11                   1           0
          3           1 01-DEC-01 20-NOV-11         0           0
          4           1 02-DEC-01 20-NOV-11         0           0
          5           1 02-DEC-01 20-NOV-11         0           0
         22           9 03-DEC-11                   1           0
         23           9 03-DEC-11                   1           1
         11           9 04-DEC-11                   1           0
         12           9 04-DEC-11                   1           0
```

```
        38         10 03-DEC-11                  1          0
        39         10 03-DEC-11                  1          1
        24          5 03-DEC-11                  1          0
        25          5 03-DEC-11                  1          0
        26          5 04-DEC-11                  1          0
        27          5 04-DEC-11                  1          0
        32          5 03-DEC-11                  1          0
        28          5 03-DEC-11                  1          0
        37          5 03-DEC-11                  1          0

EMPLOYEE_ID POSITION_ID DATE_ACQU DATE_REMO IS_PRIMARY IS_TRAINING
----------- ----------- --------- --------- ---------- -----------
         6           5 02-DEC-11                  1          1
         7           5 03-DEC-11                  1          0
        28           6 03-DEC-11                  1          0
        29           6 03-DEC-11                  1          0
        30           6 04-DEC-11                  1          0
        31           6 04-DEC-11                  1          0
        32           6 03-DEC-11                  1          0
        33           6 03-DEC-11                  1          0
         6           6 02-DEC-11                  1          0
         7           6 03-DEC-11                  1          0
        32           7 03-DEC-11                  1          0
        33           7 03-DEC-11                  1          1
        34           7 04-DEC-11                  1          0
        35           7 04-DEC-11                  1          0
        30           7 04-DEC-11                  1          0
        31           7 04-DEC-11                  1          0
         6           7 02-DEC-11                  1          1
         7           7 03-DEC-11                  1          0
        36           8 03-DEC-11                  1          1
        37           8 03-DEC-11                  1          0
        34           8 04-DEC-11                  1          0
        35           8 04-DEC-11                  1          0
        30           8 04-DEC-11                  1          0
        31           8 04-DEC-11                  1          0
         6           8 02-DEC-11                  1          0
         7           8 03-DEC-11                  1          0

63 rows selected.
CS342 SQL> DESC BAJS_HOLIDAY;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- -------
 HDATE                                                                   NOT NULL VARCHAR2(10)
 NAME                                                                             VARCHAR2(20)
 TYPE                                                                             VARCHAR2(5)

CS342 SQL> SELECT * FROM BAJS_HOLIDAY;

HDATE      NAME                TYPE
---------- ------------------- -----
12/25      Christmas           full
12/24      Christmas Eve       half
12/31      New Years Eve       half
01/01      New Years Day       half
10/31      Halloween           half
11/27      Thanksgiving        half
06/07      Easter              full

7 rows selected
```

```
CS342 SQL> DESC BAJS_INGREDIENT;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- --------
 ID                                                                      NOT NULL NUMBER(5)
 NAME                                                                             VARCHAR2(40)

CS342 SQL> SELECT * FROM BAJS_INGREDIENT;

        ID NAME
---------- ----------------------------------------
         1 potato
         2 tomato
         3 broccoli
         4 corn
         5 rice
         6 jalapeno
         7 red onion
         8 yellow onion
         9 flour
        10 chicken
        11 pork
        12 beef
        13 bacon
        14 turkey
        15 tuna
        16 chedder cheese
        17 swiss cheese
        18 provolone cheese
        19 yellow mustard
        20 honey mustard

20 rows selected.
CS342 SQL> DESC BAJS_IS_SCHEDULED_FOR;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- --------
 EMPLOYEE_ID                                                             NOT NULL NUMBER(5)
 SHIFT_ID                                                                NOT NULL NUMBER(5)
 ON_DATE                                                                 NOT NULL DATE

CS342 SQL> SELECT * FROM BAJS_IS_SCHEDULED_FOR;

EMPLOYEE_ID   SHIFT_ID ON_DATE
----------- ---------- ---------
          1          1 02-NOV-15
          1          1 03-NOV-15
          1          1 06-NOV-15
          1         10 04-NOV-15
          2          2 02-NOV-15
          2          2 03-NOV-15
          2         10 05-NOV-15
          3          1 05-NOV-15
          3          1 07-NOV-15
          3          1 08-NOV-15
          3         10 02-NOV-15
          3         10 03-NOV-15
          4          1 04-NOV-15
          4          2 05-NOV-15
          4          2 07-NOV-15
          4          2 08-NOV-15
          4         10 06-NOV-15
          5          2 04-NOV-15
          5          2 06-NOV-15
          5         10 07-NOV-15
          5         10 08-NOV-15
          6         15 02-NOV-15
          6         15 03-NOV-15
          6         15 04-NOV-15
          6         15 05-NOV-15
          7         15 06-NOV-15
          7         15 07-NOV-15
          7         15 08-NOV-15
          7         20 03-NOV-15
```

```
         8          8 02-NOV-15
         8          9 04-NOV-15
         8          9 07-NOV-15
         8          9 08-NOV-15
         8         11 03-NOV-15
         8         11 06-NOV-15
         9          8 03-NOV-15
         9          8 07-NOV-15

EMPLOYEE_ID   SHIFT_ID ON_DATE
----------- ---------- ---------
         9          8 08-NOV-15
         9          9 02-NOV-15
         9         11 04-NOV-15
        10          9 03-NOV-15
        10         11 02-NOV-15
        10         11 05-NOV-15
        11          8 04-NOV-15
        11          9 05-NOV-15
        11          9 06-NOV-15
        11         11 07-NOV-15
        11         11 08-NOV-15
        11         23 02-NOV-15
        11         24 06-NOV-15
        12          8 05-NOV-15
        12          8 06-NOV-15
        12         24 03-NOV-15
        12         24 04-NOV-15
        13          3 02-NOV-15
        13          4 04-NOV-15
        13          4 07-NOV-15
        13          4 08-NOV-15
        14          3 03-NOV-15
        14          3 05-NOV-15
        14          3 07-NOV-15
        14          3 08-NOV-15
        14          4 02-NOV-15
        14          5 04-NOV-15
        15          4 03-NOV-15
        15          4 05-NOV-15
        15          5 02-NOV-15
        15         14 06-NOV-15
        16          3 04-NOV-15
        16          6 02-NOV-15
        16         13 06-NOV-15
        17          5 03-NOV-15
        17          5 05-NOV-15
        17         12 02-NOV-15

EMPLOYEE_ID   SHIFT_ID ON_DATE
----------- ---------- ---------
        17         12 06-NOV-15
        17         13 07-NOV-15
        17         13 08-NOV-15
        18          6 03-NOV-15
        18          6 05-NOV-15
        18          6 06-NOV-15
        18         12 07-NOV-15
        18         12 08-NOV-15
        18         13 02-NOV-15
        18         14 04-NOV-15
        19          5 06-NOV-15
        19         12 03-NOV-15
        19         12 05-NOV-15
        19         13 04-NOV-15
        19         14 02-NOV-15
        19         14 07-NOV-15
        19         14 08-NOV-15
        20          4 06-NOV-15
        20          5 07-NOV-15
        20          5 08-NOV-15
        20          6 04-NOV-15
```

```
         20         13 03-NOV-15
         20         13 05-NOV-15
         21          3 06-NOV-15
         21          6 07-NOV-15
         21          6 08-NOV-15
         21         12 04-NOV-15
         21         14 03-NOV-15
         21         14 05-NOV-15
         22         23 04-NOV-15
         22         23 05-NOV-15
         22         23 07-NOV-15
         22         23 08-NOV-15
         22         24 02-NOV-15
         23         23 03-NOV-15
         23         23 06-NOV-15
         23         24 05-NOV-15
EMPLOYEE_ID   SHIFT_ID ON_DATE
----------- ---------- ---------
         23         24 07-NOV-15
         23         24 08-NOV-15
         24         16 02-NOV-15
         24         16 03-NOV-15
         25         16 04-NOV-15
         25         16 05-NOV-15
         26         16 06-NOV-15
         27         16 07-NOV-15
         27         16 08-NOV-15
         28         17 02-NOV-15
         28         17 06-NOV-15
         28         20 04-NOV-15
         28         20 07-NOV-15
         28         20 08-NOV-15
         29         17 04-NOV-15
         29         20 02-NOV-15
         30         17 03-NOV-15
         30         21 05-NOV-15
         30         21 06-NOV-15
         30         21 07-NOV-15
         30         21 08-NOV-15
         31         17 05-NOV-15
         31         19 02-NOV-15
         31         19 04-NOV-15
         31         19 07-NOV-15
         31         19 08-NOV-15
         32         17 07-NOV-15
         32         17 08-NOV-15
         32         18 02-NOV-15
         32         18 03-NOV-15
         32         19 05-NOV-15
         32         19 06-NOV-15
         33         18 04-NOV-15
         33         18 07-NOV-15
         33         18 08-NOV-15
         33         19 03-NOV-15
         33         20 05-NOV-15
EMPLOYEE_ID   SHIFT_ID ON_DATE
----------- ---------- ---------
         33         20 06-NOV-15
         34         18 06-NOV-15
         34         21 02-NOV-15
         34         21 03-NOV-15
         34         21 04-NOV-15
         34         22 05-NOV-15
         35         18 05-NOV-15
         36         22 03-NOV-15
         36         22 04-NOV-15
         36         22 06-NOV-15
         36         22 07-NOV-15
         36         22 08-NOV-15
         36         25 02-NOV-15
         37         22 02-NOV-15
         37         25 03-NOV-15
```

```
         37         25 04-NOV-15
         37         25 05-NOV-15
         37         25 06-NOV-15
         37         25 07-NOV-15
         37         25 08-NOV-15
         38          7 02-NOV-15
         38          7 03-NOV-15
         38          7 04-NOV-15
         39          7 05-NOV-15
         39          7 06-NOV-15
         39          7 07-NOV-15
         39          7 08-NOV-15
175 rows selected.

CS342 SQL> DESC BAJS_MENU_ITEM;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- -------
 ID                                                                      NOT NULL NUMBER(5)
 NAME                                                                             VARCHAR2(30)
 TYPE                                                                             VARCHAR2(30)
 PRICE                                                                            NUMBER(4,2)
 PHOTO                                                                            RAW(1024)

CS342 SQL> SELECT * FROM BAJS_MENU_ITEM;
        ID NAME                           TYPE                              PRICE PHOTO
---------- ------------------------------ ------------------------------ ---------- ----------
         1 Roast Turkey Breast            cold sandwich                        7.95
         2 California Chicken Breast      hot sandwich                         7.95
         3 Tuna Melt                      hot sandwich                         7.95
         4 Turkey Bacon Melt              hot sandwich                         7.95
         5 Classic Ruben                  hot sandwich                         7.95
         6 California Pastrami            hot sandwich                         7.95
         7 Country Potato                 cream soup                           5.95
         8 Tomato Bisque                  cream soup                           5.95
         9 BroccoliCheese                 cream soup                           5.95
        10 Corn Chowder                   cream soup                           5.95
10 rows selected.

CS342 SQL> DESC BAJS_ORDERS;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- -------
 EMPLOYEE_ID                                                                      NUMBER(5)
 INGREDIENT_ID                                                                    NUMBER(5)
 ON_DATE                                                                          DATE
 QUANTITY                                                                         NUMBER(3)

CS342 SQL> SELECT * FROM BAJS_ORDERS;

EMPLOYEE_ID INGREDIENT_ID ON_DATE     QUANTITY
----------- ------------- --------- ----------
          4             1 01-MAY-15         10
          7             2 01-MAY-15         11
         10             3 01-MAY-15         12
         14             4 01-MAY-15         13
         17             5 01-MAY-15         14
         20             6 01-MAY-15         15
          4             7 02-MAY-15         10
          7             8 02-MAY-15         11
         10             9 02-MAY-15         12
         14            10 02-MAY-15         13
         17             1 02-MAY-15         14
         20             2 02-MAY-15         15
         10             3 03-MAY-15         12
         14             4 03-MAY-15         13
         17             5 03-MAY-15         14
         20             6 03-MAY-15         15
          4             7 04-MAY-15         10
          7             8 04-MAY-15         11
         10             9 04-MAY-15         12
         14            10 04-MAY-15         13
         17             1 04-MAY-15         14
         20             2 04-MAY-15         15
```

```
         10            3 05-MAY-15        12
         14            4 05-MAY-15        13
         17            5 05-MAY-15        14
         20            6 05-MAY-15        15
          4            7 06-MAY-15        10
          7            8 06-MAY-15        11
         10            9 06-MAY-15        12
         14           10 06-MAY-15        13
         17            1 06-MAY-15        14
         20            2 06-MAY-15        15
         10            3 07-MAY-15        12
         14            4 07-MAY-15        13
         17            5 07-MAY-15        14
         20            6 07-MAY-15        15
          4            7 08-MAY-15        10

EMPLOYEE_ID INGREDIENT_ID ON_DATE     QUANTITY
----------- ------------- --------- ----------
          7            8 08-MAY-15        11
         10            9 08-MAY-15        12
         14           10 08-MAY-15        13
         17            1 08-MAY-15        14
         20            2 08-MAY-15        15
         10            3 09-MAY-15        12
         14            4 09-MAY-15        13
         17            5 09-MAY-15        14
         20            6 09-MAY-15        15
          4            7 10-MAY-15        10
          7            8 10-MAY-15        11
         10            9 10-MAY-15        12
         14           10 10-MAY-15        13
         17            1 10-MAY-15        14
         20            2 10-MAY-15        15
         10            3 11-MAY-15        12
         14            4 11-MAY-15        13
         17            5 11-MAY-15        14
         20            6 11-MAY-15        15
          4            7 12-MAY-15        10
          7            8 12-MAY-15        11
         10            9 12-MAY-15        12
         14           10 12-MAY-15        13

60 rows selected.
CS342 SQL> DESC BAJS_POSITION;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- -------
 ID                                                                      NOT NULL NUMBER(5)
 TITLE                                                                            VARCHAR2(20)
 LOCATION                                                                         VARCHAR2(10)

CS342 SQL> SELECT * FROM BAJS_POSITION;

        ID TITLE               LOCATION
---------- ------------------- ----------
         1 Supervisor          front
         2 Cashier             front
         3 Cold Prep           front
        10 Busser              front
         4 Kitchen Supervisor  kitchen
         5 Bakery              kitchen
         6 Grill               kitchen
         7 Salads              kitchen
         8 Other Kitchen       kitchen
         9 Janitor             kitchen

10 rows selected.
```

```
CS342 SQL> DESC BAJS_REQUESTS_VACATION;
 Name                                                      Null?    Type
 ----------------------------------------------------------------- --------
 REQUESTED_BY                                              NOT NULL NUMBER(5)
 RESPONDED_BY                                              NOT NULL NUMBER(5)
 IS_APPROVED                                                        NUMBER(1)
 START_DATE_TIME                                           NOT NULL TIMESTAMP(6)
 END_DATE_TIME                                                      TIMESTAMP(6)

CS342 SQL> SELECT * FROM BAJS_REQUESTS_VACATION;

REQUESTED_BY RESPONDED_BY IS_APPROVED START_DATE_TIME END_DATE_TIME
------------ ------------ ----------- ---------------------------------------

           1            4           1 10-SEP-02 02.10.10.123000 PM 10-SEP-02 02.20.10.123000 PM

           2            7           0 10-SEP-02 02.10.10.123000 PM 10-SEP-02 02.20.10.123000 PM

           3           10           0 10-SEP-02 02.10.10.123000 PM 10-SEP-02 02.20.10.123000 PM

           4           14           1 11-SEP-02 02.10.10.123000 PM 11-SEP-02 02.20.10.123000 PM

           5           17           0 11-SEP-02 02.10.10.123000 PM 11-SEP-02 02.20.10.123000 PM

           6           20           0 11-SEP-02 02.10.10.123000 PM 11-SEP-02 02.20.10.123000 PM

           7            4           1 12-SEP-02 02.10.10.123000 PM 12-SEP-02 02.20.10.123000 PM

           8            7           0 12-SEP-02 02.10.10.123000 PM 12-SEP-02 02.20.10.123000 PM

           9           10           0 12-SEP-02 02.10.10.123000 PM 12-SEP-02 02.20.10.123000 PM

          10            4           1 13-SEP-02 02.10.10.123000 PM 13-SEP-02 02.20.10.123000 PM

          11            7           0 13-SEP-02 02.10.10.123000 PM 13-SEP-02 02.20.10.123000 PM

          12           10           0 13-SEP-02 02.10.10.123000 PM 13-SEP-02 02.20.10.123000 PM

          13           14           1 14-SEP-02 02.10.10.123000 PM 14-SEP-02 02.20.10.123000 PM

          14           17           0 14-SEP-02 02.10.10.123000 PM 14-SEP-02 02.20.10.123000 PM

          15           20           0 14-SEP-02 02.10.10.123000 PM 14-SEP-02 02.20.10.123000 PM

          16            4           1 15-SEP-02 02.10.10.123000 PM 15-SEP-02 02.20.10.123000 PM

          17            7           0 15-SEP-02 02.10.10.123000 PM 15-SEP-02 02.20.10.123000 PM

          18           10           0 15-SEP-02 02.10.10.123000 PM 15-SEP-02 02.20.10.123000 PM

          19           14           1 16-SEP-02 02.10.10.123000 PM 16-SEP-02 02.20.10.123000 PM

          20           17           0 16-SEP-02 02.10.10.123000 PM 16-SEP-02 02.20.10.123000 PM

           1            4           1 01-OCT-02 02.10.10.123000 PM 02-OCT-02 02.10.10.123000 PM

           2            7           0 03-OCT-02 02.10.10.123000 PM 04-OCT-02 02.10.10.123000 PM

           3           10           0 05-OCT-02 02.10.10.123000 PM 06-OCT-02 02.10.10.123000 PM

           4           14           1 07-OCT-02 02.10.10.123000 PM 08-OCT-02 02.10.10.123000 PM

REQUESTED_BY RESPONDED_BY IS_APPROVED START_DATE_TIME END_DATE_TIME
------------ ------------ ---------------------------------------

           5           17           0 09-OCT-02 02.10.10.123000 PM 10-OCT-02 02.10.10.123000 PM

           6           20           0 11-OCT-02 02.10.10.123000 PM 12-OCT-02 02.10.10.123000 PM

           7            4           1 13-OCT-02 02.10.10.123000 PM 14-OCT-02 02.10.10.123000 PM
```

```
         8          7          0 15-OCT-02 02.10.10.123000 PM 16-OCT-02 02.10.10.123000 PM
         9         10          0 17-OCT-02 02.10.10.123000 PM 18-OCT-02 02.10.10.123000 PM
        10          4          1 19-OCT-02 02.10.10.123000 PM 20-OCT-02 02.10.10.123000 PM
        11          7          0 21-OCT-02 02.10.10.123000 PM 22-OCT-02 02.10.10.123000 PM
        12         10          0 23-OCT-02 02.10.10.123000 PM 24-OCT-02 02.10.10.123000 PM
        13         14          1 25-OCT-02 02.10.10.123000 PM 26-OCT-02 02.10.10.123000 PM
        14         17          0 27-OCT-02 02.10.10.123000 PM 28-OCT-02 02.10.10.123000 PM
        15         20          0 29-OCT-02 02.10.10.123000 PM 30-OCT-02 02.10.10.123000 PM

REQUESTED_BY RESPONDED_BY IS_APPROVED START_DATE_TIME END_DATE_TIME
------------ ------------ ----------- ------------------------------------
        16          4          1 01-NOV-02 02.10.10.123000 PM02-NOV-02 02.10.10.123000 PM
        17          7          0 03-NOV-02 02.10.10.123000 PM04-NOV-02 02.10.10.123000 PM
        18         10          0 05-NOV-02 02.10.10.123000 PM06-NOV-02 02.10.10.123000 PM
        19         14          1 07-NOV-02 02.10.10.123000 PM08-NOV-02 02.10.10.123000 PM
        20         17          0 09-NOV-02 02.10.10.123000 PM10-NOV-02 02.10.10.123000 PM
         1          4          1 10-NOV-02 02.10.10.123000 PM11-NOV-02 02.10.10.123000 PM
         2          7          0 11-NOV-02 02.10.10.123000 PM12-NOV-02 02.10.10.123000 PM
         3         10          0 12-NOV-02 02.10.10.123000 PM13-NOV-02 02.10.10.123000 PM
         4         14          1 13-NOV-02 02.10.10.123000 PM14-NOV-02 02.10.10.123000 PM
         5         17          0 14-NOV-02 02.10.10.123000 PM15-NOV-02 02.10.10.123000 PM
         6         20          0 15-NOV-02 02.10.10.123000 PM16-NOV-02 02.10.10.123000 PM
         7          4          1 16-NOV-02 02.10.10.123000 PM17-NOV-02 02.10.10.123000 PM
         8          7          0 17-NOV-02 02.10.10.123000 PM 18-NOV-02 02.10.10.123000 PM
         9         10          0 18-NOV-02 02.10.10.123000 PM 19-NOV-02 02.10.10.123000 PM
        10          2          1 19-NOV-02 02.10.10.123000 PM 20-NOV-02 02.10.10.123000 PM
        11          3          0 21-NOV-02 02.10.10.123000 PM 22-NOV-02 02.10.10.123000 PM
        12          4          0 22-NOV-02 02.10.10.123000 PM 23-NOV-02 02.10.10.123000 PM
        13          5          1 23-NOV-02 02.10.10.123000 PM 24-NOV-02 02.10.10.123000 PM
        14          5          0 24-NOV-02 02.10.10.123000 PM 25-NOV-02 02.10.10.123000 PM
        15          5          0 28-NOV-02 02.10.10.123000 PM 29-NOV-02 02.10.10.123000 PM
        16          6          1 29-NOV-02 02.10.10.123000 PM 30-NOV-02 02.10.10.123000 PM
         6          5          1 07-SEP-02 02.00.00.123000 PM 07-SEP-02 02.24.00.123000 PM
         5          2          1 26-NOV-02 02.00.00.123000 PM 27-NOV-02 02.24.00.123000 PM
         2          1          1 25-DEC-02 02.00.00.123000 PM 25-DEC-02 02.24.00.123000 PM
         1          2          1 31-DEC-02 02.15.10.123000 PM 01-JAN-02 02.12.10.123000 PM
60 rows selected.

CS342 SQL> DESC BAJS_SHIFT;
 Name                                                      Null?    Type
 ----------------------------------------------------------------- --------
```

```
 ID                                                        NOT NULL NUMBER(5)
 POSITION_ID                                                        NUMBER(5)
 TASK_NAME                                                          VARCHAR2(30)

CS342 SQL> SELECT * FROM BAJS_SHIFT;

        ID POSITION_ID TASK_NAME
---------- ----------- ------------------------------
         1           1 Opening Sup/Mgr
         2           1 Support Sup/Staff
         3           2 Front (tomatoes)
         4           2 Front (dressings)
         5           2 Front (register 1)
         6           2 Front
         7          10 Busser
         8           3 Cheese
         9           3 Meat
        10           1 Closing Supervisor
        11           3 Cold Prep
        12           2 Register 1
        13           2 Expedite
        14           2 Front/Busser
        15           4 Kitchen Sup or Staff
        16           5 Bakery
        17           6 Grill
        18           7 Salads
        25           8 Support Staff
        19           7 Kitchen Sup or Staff
        20           6 Grill
        21           7 Salads
        22           8 Support Staff
        23           9 Closing Janitor
        24           9 Opening Janitor

25 rows selected.

CS342 SQL> DESC BAJS_SOLD_IN;
 Name                                                      Null?    Type
 ----------------------------------------------------------------- --------
 MENU_ITEM_ID                                                       NUMBER(5)
 TRANSACTION_ID                                                     NUMBER(5)
 QUANTITY                                                           NUMBER(5)
CS342 SQL> SELECT * FROM BAJS_SOLD_IN;
MENU_ITEM_ID TRANSACTION_ID   QUANTITY
------------ -------------- ----------
           1              1          1
           2              1          2
           3              1          3
           4              1          1
           5              1          2
           6              1          3
           7              2          1
           8              2          2
           9              2          1
          10              2          2
           1              2          1
           2              2          2
           3              3          3
           4              3          1
           5              3          2
           6              3          3
           7              3          1
           8              3          2
           9              4          1
          10              4          2
           1              4          1
           2              4          2
           3              4          3
           4              4          1
           5              5          2
           6              5          3
           7              5          1
```

```
           8             5          2
           9             5          1
          10             5          2
           1             6          1
           2             6          2
           3             6          3
           4             6          1
           5             6          2
           6             6          3
           7             7          1
MENU_ITEM_ID TRANSACTION_ID   QUANTITY
------------ -------------- ----------
           8             7          2
           9             7          1
          10             7          2
           1             7          1
           2             7          2
           3             8          3
           4             8          1
           5             8          2
           6             8          3
           7             8          1
           8             8          2
           9             9          1
          10             9          2
           1             9          1
           2             9          2
           3             9          3
           4             9          1
           5            10          2
           6            10          3
           7            10          1
           8            10          2
           9            10          1
          10            10          2
60 rows selected.

CS342 SQL> DESC BAJS_SUPPLIER;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- -------
 ID                                                                      NOT NULL NUMBER(5)
 TITLE                                                                            VARCHAR2(30)
 DELIVERY_DAYS                                                                    VARCHAR2(15)

CS342 SQL> SELECT * FROM BAJS_SUPPLIER;

        ID TITLE                          DELIVERY_DAYS
---------- ------------------------------ ---------------
         1 Pepsi                          m
         2 Sysco                          t r s
         3 Alpha                          m t w r f s
         4 Boars Head                     m
         5 Gellasos                       m w r
         6 Pyreneese                      m t w r f s
         7 Beta                           f s
         8 Kappa                          m w s
         9 Gamma                          t f
        10 Delta                          w r

10 rows selected.
```

```
CS342 SQL> DESC BAJS_TRANSACTION;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- --------
 ID                                                                      NOT NULL NUMBER(5)
 ON_DATE                                                                          DATE

CS342 SQL> SELECT * FROM BAJS_TRANSACTION;

        ID ON_DATE
---------- ---------
         1 01-DEC-14
         2 01-DEC-14
         3 01-DEC-14
         4 01-DEC-14
         5 01-DEC-14
         6 02-DEC-14
         7 02-DEC-14
         8 02-DEC-14
         9 02-DEC-14
        10 02-DEC-14
        11 01-DEC-14
        12 01-DEC-14
        13 01-DEC-14
        14 01-DEC-14
        15 01-DEC-14
        16 02-DEC-14
        17 02-DEC-14
        18 02-DEC-14
        19 02-DEC-14
        20 02-DEC-14
        21 01-DEC-14
        22 01-DEC-14
        23 01-DEC-14
        24 01-DEC-14
        25 01-DEC-14
        26 02-DEC-14
        27 02-DEC-14
        28 02-DEC-14
        29 02-DEC-14
        30 02-DEC-14
        31 01-DEC-14
        32 01-DEC-14
        33 01-DEC-14
        34 01-DEC-14
        35 01-DEC-14
        36 02-DEC-14
        37 02-DEC-14
        ID ON_DATE
---------- ---------
        38 02-DEC-14
        39 02-DEC-14
        40 02-DEC-14
        41 01-DEC-14
        42 01-DEC-14
        43 01-DEC-14
        44 01-DEC-14
        45 01-DEC-14
        46 02-DEC-14
        47 02-DEC-14
        48 02-DEC-14
        49 02-DEC-14
        50 02-DEC-14
        51 01-DEC-14
        52 01-DEC-14
        53 01-DEC-14
        54 01-DEC-14
        55 01-DEC-14
        56 02-DEC-14
        57 02-DEC-14
        58 02-DEC-14
        59 02-DEC-14
        60 02-DEC-14
```

```
60 rows selected.

CS342 SQL> DESC BAJS_USED_IN;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- --------
 MENU_ITEM_ID                                                                     NUMBER(5)
 INGREDIENT_ID                                                                    NUMBER(5)
 QUANTITY                                                                         NUMBER(5)

CS342 SQL> SELECT * FROM BAJS_USED_IN;

MENU_ITEM_ID INGREDIENT_ID   QUANTITY
------------ ------------- ----------
           1             1          5
           1             2          6
           1             3          5
           1             4          6
           1             5          5
           1             6          6
           2             7          5
           2             8          6
           2             9          5
           2            10          6
           2             1          5
           2             2          6
           3             3          5
           3             4          6
           3             5          5
           3             6          6
           3             7          5
           3             8          6
           4             9          5
           4            10          6
           4             1          5
           4             2          6
           4             3          5
           4             4          6
           5             5          5
           5             6          6
           5             7          5
           5             8          6
           5             9          5
           5            10          6
           6             1          5
           6             2          6
           6             3          5
           6             4          6
           6             5          5
           6             6          6
           7             7          5

MENU_ITEM_ID INGREDIENT_ID   QUANTITY
------------ ------------- ----------
           7             8          6
           7             9          5
           7            10          6
           7             1          5
           7             2          6
           8             3          5
           8             4          6
           8             5          5
           8             6          6
           8             7          5
           8             8          6
           9             9          5
           9            10          6
           9             1          5
           9             2          6
           9             3          5
           9             4          6
          10             5          5
```

```
         10           6         6
         10           7         5
         10           8         6
         10           9         5
         10          10         6

60 rows selected.
CS342 SQL> DESC BAJS_USER;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- --------
 EMPLOYEE_ID                                                                      NUMBER(5)
 EMAIL                                                                            VARCHAR2(30)
 PASSWORD                                                                         VARCHAR2(30)
 API_TOKEN                                                                        VARCHAR2(30)

CS342 SQL> SELECT * FROM BAJS_USER;

EMPLOYEE_ID EMAIL                          PASSWORD                      API_TOKEN
----------- ------------------------------ ----------------------------- ------------------------------
          1 emp1@example.com               password                      asdn4%og23blsfqa~>8f&;aeri
          2 emp2@example.com               password                      bsdn4%og23blsfqa~>8f&;aeri
          3 emp3@example.com               password                      csdn4%og23blsfqa~>8f&;aeri
          4 emp4@example.com               password                      dsdn4%og23blsfqa~>8f&;aeri
          5 emp5@example.com               password                      esdn4%og23blsfqa~>8f&;aeri
          6 emp6@example.com               password                      fsdn4%og23blsfqa~>8f&;aeri
          7 emp7@example.com               password                      gsdn4%og23blsfqa~>8f&;aeri
          8 emp8@example.com               password                      hsdn4%og23blsfqa~>8f&;aeri
          9 emp9@example.com               password                      isdn4%og23blsfqa~>8f&;aeri
         10 emp10@example.com              password                      jsdn4%og23blsfqa~>8f&;aeri
         11 emp11@example.com              password                      ksdn4%og23blsfqa~>8f&;aeri
         12 emp12@example.com              password                      lsdn4%og23blsfqa~>8f&;aeri
         13 emp13@example.com              password                      msdn4%og23blsfqa~>8f&;aeri
         14 emp14@example.com              password                      nsdn4%og23blsfqa~>8f&;aeri
         15 emp15@example.com              password                      osdn4%og23blsfqa~>8f&;aeri
         16 emp16@example.com              password                      psdn4%og23blsfqa~>8f&;aeri
         17 emp17@example.com              password                      qsdn4%og23blsfqa~>8f&;aeri
         18 emp18@example.com              password                      rsdn4%og23blsfqa~>8f&;aeri
         19 emp19@example.com              password                      ssdn4%og23blsfqa~>8f&;aeri
         20 emp20@example.com              password                      tsdn4%og23blsfqa~>8f&;aeri

20 rows selected.
CS342 SQL> DESC BAJS_WEEKDAY_HOURS;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- --------
 SHIFT_ID                                                                         NUMBER(5)
 START_HOUR                                                                       NUMBER(4)
 END_HOUR                                                                         NUMBER(4)

CS342 SQL> SELECT * FROM BAJS_WEEKDAY_HOURS;

  SHIFT_ID START_HOUR  END_HOUR
---------- ---------- ----------
         2        700       1500
         4        800       1530
         5       1100       1600
         6        900       1400
         7       1115       1515
         8        630       1430
         9        630       1430
        10       1200       2030
        11       1300       2030
        12       1530       2030
        13       1530       2030
        14       1600       2000
        15        700       1500
        16        700       1500
        17        700       1500
        18        700       1500
        25        700       1500
        19       1330       2030
        20       1430       2030
        21       1500       2030
```

```
        22       1500       2030
        23        730       1500
        24       1500       2030
23 rows selected.

CS342 SQL> DESC BAJS_WEEKEND_HOURS;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- -------
 SHIFT_ID                                                                         NUMBER(5)
 START_HOUR                                                                       NUMBER(4)
 END_HOUR                                                                         NUMBER(4)

CS342 SQL> SELECT * FROM BAJS_WEEKEND_HOURS;

  SHIFT_ID START_HOUR  END_HOUR
---------- ---------- ----------
         1        700       1500
         3        800       1500
         4        800       1530
         5       1100       1600
         6        900       1400
         7       1115       1515
         8        700       1500
         9        700       1500
        10       1200       2030
        11       1300       2030
        12       1500       2030
        13       1500       2030
        14       1600       2000
        15        700       1500
        16        700       1500
        17        700       1500
        18        700       1500
        25        700       1500
        19       1330       2030
        20       1430       2030
        21       1500       2030
        22       1500       2030
        23        730       1500
        24       1500       2030
24 rows selected.
```

Section 3.5 Example Queries in SQL

This section describes the sample queries from Phase 2 in the SQL language. Additional queries have also been included to demonstrate SQL functionality that could not be shown within the queries from Phase 2.

1. Find all employees who have been scheduled to work as a closing janitor.

```
SELECT DISTINCT eid
FROM(
  (
        SELECT i.shift_id AS sid, i.employee_id AS eid
        FROM Is_Scheduled_For i
  )
  NATURAL_JOIN
  (
        SELECT s.task_name AS tname, s.id AS sid
        FROM Shift s
        WHERE s.tname = 'Closing Janitor'
  )
)
/
```

```
 CS342 SQL> @q1

       SID        EID TNAME
---------- ---------- ------------------------------
 23         11 Closing Janitor
 23         22 Closing Janitor
 23         22 Closing Janitor
 23         22 Closing Janitor
 23         22 Closing Janitor
 23         23 Closing Janitor
 23         23 Closing Janitor

 7 rows selected.
```

2. Find all employees who can work the tasked named register1 on Mondays. Include employees still training for the task.

```
SELECT eid, name
FROM (
  (
        SELECT s.id AS sid, s.task_name AS tname, s.position_id AS pid
        FROM bajs_shift s
        WHERE s.task_name = 'Register 1'
  )
  NATURAL JOIN
  (
        SELECT w.shift_id AS sid, w.start_hour, w.start_hour AS wstartt, w.end_hour AS wendt
        FROM bajs_weekday_hours w
  )
  NATURAL JOIN
  (
        SELECT e.eid, e.pid, e.name
        FROM bajs_std_emp e
  )
  NATURAL JOIN
  (
        SELECT a.employee_id AS eid, a.day, a.startt AS astartt, a.endt AS aendt
        FROM bajs_availability a
        WHERE day = 'mon'
  )
)
```

```
      WHERE wstartt >= astartt AND wendt <= aendt
      /

    CS342 SQL> @q2

           EID NAME
    ---------- ----------
        15 Jules
        21 Udo
        17 Sammie
        18 Hadley
        19 Chike
        16 Li
        20 Udo
        13 Maria

    8 rows selected.
```

3. Find all cashiers available from 14:00-21:00 on all weekdays

```
SELECT DISTINCT *
FROM (
        (
                SELECT h.eid, h.name
                FROM bajs_std_emp h
                WHERE h.title = 'Cashier'
        )
        NATURAL JOIN
        (
                SELECT a.employee_id AS eid, a.day
                FROM bajs_availability a
                WHERE a.startt <= 1400 AND
                a.endt >= 2100 AND
                (a.day='mon' OR a.day='tue' OR a.day='wed' OR a.day='thu' OR a.day='fri')
        )
)
ORDER BY day
/
CS342 SQL> @q3

       EID NAME       DAY
---------- ---------- --------
        19 Chike      fri
        18 Hadley     fri
        15 Jules      fri
        16 Li         fri
        13 Maria      fri
        17 Sammie     fri
        20 Udo        fri
        21 Udo        fri
        19 Chike      mon
        18 Hadley     mon
        15 Jules      mon
        16 Li         mon
        13 Maria      mon
        17 Sammie     mon
        20 Udo        mon
        21 Udo        mon
        19 Chike      thu
        18 Hadley     thu
        15 Jules      thu
        16 Li         thu
        13 Maria      thu
        17 Sammie     thu
        20 Udo        thu
        21 Udo        thu
        19 Chike      tue
        18 Hadley     tue
        15 Jules      tue
        16 Li         tue
        13 Maria      tue
```

```
        17 Sammie     tue
        20 Udo        tue
        21 Udo        tue
        19 Chike      wed
        18 Hadley     wed
        15 Jules      wed
        16 Li         wed
        13 Maria      wed

       EID NAME       DAY
---------- ---------- --------
        17 Sammie     wed
        20 Udo        wed
        21 Udo        wed

40 rows selected.
```

4. Find employees who can work every position in the kitchen, where they are not still training for any of them

```
SELECT *
FROM (
  (
        SELECT p.id AS pid
        FROM bajs_position p
        WHERE p.location = 'kitchen'
  )
  NATURAL JOIN
  (
        SELECT *
        FROM bajs_std_emp
  )
)
ORDER BY name
/
```

```
CS342 SQL> @q4

       PID       EID TITLE              NAME
---------- ---------- ------------------ ----------
         5         28 Bakery             Ade
         6         28 Grill              Ade
         6         33 Grill              Aminta
         8         34 Other Kitchen      Devraj
         7         34 Salads             Devraj
         6         29 Grill              Egill
         6         32 Grill              Gina
         7         32 Salads             Gina
         5         32 Bakery             Gina
         8          6 Other Kitchen      Kamala
         6          6 Grill              Kamala
         8         31 Other Kitchen      Kasey
         6         31 Grill              Kasey
         7         31 Salads             Kasey
         7          7 Salads             Lei
         4          7 Kitchen Supervisor  Lei
         5          7 Bakery             Lei
         8          7 Other Kitchen      Lei
         6          7 Grill              Lei
         7         30 Salads             Marcelli
         6         30 Grill              Marcelli
         8         30 Other Kitchen      Marcelli
         9         11 Janitor            Mavuto
         8         35 Other Kitchen      Nikolas
         7         35 Salads             Nikolas
         5         27 Bakery             Paora
         5         25 Bakery             Shyama
         9         12 Janitor            Steph
         9         22 Janitor            Udo
         5         26 Bakery             Ulyses
```

```
         5         37 Bakery             Vinzent
         8         37 Other Kitchen      Vinzent
         5         24 Bakery             Zephyrus

33 rows selected.
```

5. find all supervisors who can work the earliest supervisor shift on Saturday or Sunday.

```
CREATE TABLE bajs_super_shifts AS
(
SELECT *
FROM (
        (
              SELECT p.id AS pid
              FROM bajs_position p
              WHERE p.title = 'Supervisor'
        )
        NATURAL JOIN
        (
              SELECT s.id AS sid, s.position_id AS pid
              FROM bajs_shift s
        )
        )
        NATURAL JOIN
        (
              SELECT w.shift_id AS sid, w.start_hour AS wstartt, w.end_hour AS wendt
              FROM bajs_weekend_hours w
        )
);

SELECT eid, name, day, astartt AS avail_start, aendt AS avail_end, sid, wstartt    AS shift_start, wendt
AS shift_end
        FROM (
                (
                        SELECT *
                        FROM bajs_std_emp e
                        WHERE e.title = 'Supervisor'
                )
                NATURAL JOIN
                (
                      SELECT a.employee_id AS eid, a.day, a.startt AS astartt, a.endt AS aendt
                      FROM bajs_availability a
                      WHERE day = 'sat' OR day = 'sun'
                )
        )
       NATURAL JOIN
        (
              SELECT *
              FROM bajs_super_shifts
              MINUS
              SELECT s1.*
              FROM (
              bajs_super_shifts s1
              JOIN
bajs_super_shifts s2
              ON s1.wstartt > s2.wstartt
        )
)
WHERE astartt <= wstart AND aendt >= wendt;

DROP TABLE bajs_super_shifts;
PURGE recyclebin;


CS342 SQL> @q5

Table created.
```

```
       EID NAME       DAY       AVAIL_START  AVAIL_END            SID SHIFT_START  SHIFT_END
---------- ---------- -------- ----------- ---------- ---------- ----------- ----------
         1 John       sat             600       2100          1         700       1500
         2 Jane       sat             600       2100          1         700       1500
         4 Susan      sat             600       2100          1         700       1500
         5 Chris      sat             600       2100          1         700       1500
         1 John       sun             600       2100          1         700       1500
         2 Jane       sun             600       2100          1         700       1500
         4 Susan      sun             600       2100          1         700       1500
         5 Chris      sun             600       2100          1         700       1500

8 rows selected.


Table dropped.
```

6. List the employees who have the second longest request for vacation

```
CREATE TABLE bajs_result1 AS
(   SELECT *
    FROM bajs_requests_vacation
    MINUS
    (   SELECT *
        FROM bajs_requests_vacation
        MINUS
        SELECT DISTINCT r1.*
        FROM bajs_requests_vacation r1, bajs_requests_vacation r2
        WHERE r1.start_date_time > r2.start_date_time
    )
);

SELECT id AS rid, requested_by AS eid, start_date_time
FROM
(
    SELECT *
    FROM bajs_result1 r3
    MINUS
    (   SELECT distinct r1.*
        FROM bajs_result1 r1, bajs_result1 r2
        WHERE r1.start_date_time > r2.start_date_time
    )
);

DROP TABLE bajs_result1;
PURGE RECYCLEBIN;
```

```
    RID       EID START_DATE_TIME
--------- --------- ---------------------------------------------------------------------------
      1         1 10-SEP-02 02.10.10.123000 PM
      2         2 10-SEP-02 02.10.10.123000 PM
      3         3 10-SEP-02 02.10.10.123000 PM
```

7. List the possible shifts that “John Doe” can work, given his availability and training

```
SELECT *
from
(
    SELECT unique tname, astart, aend, wd_st, wd_ed, we_st, we_ed, sid
    FROM (
            (
            SELECT *
```

```
            FROM (
                ( SELECT eid, pid
                    FROM bajs_std_emp
                    WHERE name = 'John'
                )
                natural JOIN
                ( SELECT id AS eid, last_name AS lname, first_name AS fname
                    FROM  bajs_employee
                )
            )
            where lname = 'Doe' AND fname = 'John'
        )
        NATURAL JOIN
        (
            SELECT position_id AS pid, id AS sid, task_name AS tname
            from  bajs_shift
        )
        NATURAL JOIN
        (
            SELECT employee_id AS eid, startt AS astart, endt AS aend
            FROM  bajs_availability
        )
    )
    JOIN
    (
        SELECT wd.shift_id AS wd_sid, we.shift_id AS we_sid, wd.start_hour AS wd_st,
            wd.end_hour AS wd_ed, we.start_hour AS we_st, we.end_hour AS we_ed
        FROM bajs_weekday_hours wd
        FULL OUTER JOIN bajs_weekend_hours we
        ON  wd.shift_id = we.shift_id
    )
    ON sid = wd_sid OR sid = we_sid
)
where (astart <= wd_st AND aend  >= wd_ed) OR (astart <= we_st AND aend>= we_ed)

/

TNAME                          ASTART       AEND      WD_ST      WD_ED      WE_ST      WE_ED       SID
---------------------------- ---------- ---------- ---------- ---------- ---------- ---------- --------
Support Sup/Staff                 600       2100        700       1500                             2
Closing Supervisor                600       2100       1200       2030       1200       2030        10
Opening Sup/Mgr                   600       2100                             700       1500         1
```

8. List the possible shifts that “John Doe” can work, given his availability and training

```
SELECT eid
FROM (
    SELECT eid, count(pid) AS count
    FROM bajs_std_emp
    GROUP BY eid
)
WHERE count = 2
MINUS
SELECT employee1_id AS eid
FROM bajs_cannot_work_with
MINUS
SELECT employee2_id AS id
FROM bajs_cannot_work_with


       EID
----------
         6
        28
        34
        35
        37
```

9. Find sold menu items that used ingredients ordered between 05/06/15 and 05/08/15.

```
SELECT * FROM (
    (
    SELECT UNIQUE mid FROM (
        (
            SELECT ingredient_id AS iid
            FROM bajs_orders
            WHERE on_date between to_date('05/06/2015', 'mm/dd/yyyy') AND
                                  to_date('05/08/2015', 'mm/dd/yyyy')
        )
        NATURAL JOIN
        (
            SELECT ingredient_id AS iid, menu_item_id AS mid
            FROM bajs_used_in
        )
    )
    NATURAL JOIN
    (
        SELECT menu_item_id AS mid
        FROM bajs_sold_in
    )
    )
    NATURAL JOIN
    (
        SELECT m.id AS mid, name
        FROM bajs_menu_item m
    )
)
```

```
       MID NAME
---------- ------------------------------
         1 Roast Turkey Breast
         2 California Chicken Breast
         3 Tuna Melt
         4 Turkey Bacon Melt
         5 Classic Ruben
         6 California Pastrami
         7 Country Potato
         8 Tomato Bisque
         9 BroccoliCheese
        10 Corn Chowder
```

10. Find all employees who have been scheduled exactly four times for a cashier shift

```
SELECT DISTINCT eid, count
FROM (
    SELECT DISTINCT eid, count(eid) AS count
    FROM
    (
        SELECT eid, sid, day
        FROM (
            (
            SELECT employee_id AS eid, shift_id AS sid, on_date AS day
            FROM bajs_is_scheduled_for
            )
```

```
            NATURAL JOIN
            (
            SELECT id AS sid, position_id AS pid
            FROM bajs_shift
            )
        )
        NATURAL JOIN
        (
            SELECT id AS pid, title
            FROM bajs_position
            WHERE title = 'Cashier'
        )
    )
    GROUP BY eid
)
WHERE count = 4
```

```
CS342 SQL> @q10

       EID      COUNT
---------- ----------
        13          4
        15          4
```

11. Find all employees who have never been scheduled to work work on a weekend shift.

```
SELECT eid, name
FROM bajs_std_emp
NATURAL JOIN
(   SELECT eid
    FROM bajs_std_emp
    MINUS
    SELECT i.employee_id AS eid
    FROM bajs_is_scheduled_for i
    WHERE EXISTS(
        SELECT w.shift_id
        FROM bajs_weekend_hours w
        WHERE w.shift_id = i.shift_id
    )
)
/
```

```
CS342 SQL> @q11

       EID NAME
---------- ----------
         2 Jane
```

Section 3.6 Data Loader

This section describes methods for inserting, importing, and exporting data into a database.

Methods

There are several methods that can be used to load data into a table. The most basic is to insert each row one at a time into the desired table. This is done using the command:

INSERT INTO <table> VALUES(<value_for_col1>, <value_for_col2>, …, <value_for_colN>)

This command is repeated as many times as needed to insert the desired rows into the table. Eventually, every insertion method boils down to this command, and this was the method used to insert data our Schedule Database. A series of these commands were saved into separate files, one for each table. A master file was then created to run each table's insertion file in SQL/PLUS; useful for easily reloading default sample data if the tables were modified and needed to be reset.

SQL also allows table insertions using the results of queries on other tables. The format for the command is as follows:

INSERT INTO <table> <query>

This command essentially takes the results of the query and inserts the column values for each row into the first insertion method shown, then executes that command for each row. Often the results of queries are not stored in tables, since this is considered derived data that should only be acquired from queries, not saved permanently. However, a method very similar to this is how Views are created and updated.

Various import and export tools for table data are also available, which import and export data from and to different file formats. These are usually tools created by third parties, and many are available online for almost every DBMS. These tools are most efficient for loading bulk data that was generated by some other program to help quickly create an initial instance of a database for the end user. Some examples are SQL Loader, Oracle Data Pump, and SQL Express Management Studio.

Java DataLoader Program

One such data import tool was created by Dr. Huaqing Wang of California State University, Bakersfield. The tool has been given to students taking his databases class over the years to help them create their database projects, including this Fall, 2015, CMPS 342 class. This tool loads data from a file into the specified tables using a specified delimiter to separate the values. This Java DataLoader was not used in our project, so no modifications were made to it, however the program is used as follows (assuming the delimiting character is a “|”):

TABLENAME | <table_name> | <number_of_columns>
<row1_col1_value> | <row1_col2_value> | … | <row1_colN_value>
<row2_col1_value> | <row2_col2_value> | … | <row2_colN_value>
…

Phase 4 Oracle PL/SQL

Section 4.1 PL/SQL Subprograms

This section describes Oracle PL/SQL subprograms, their syntax, and the benefits of subprograms.

4.1.1 Subprograms

Oracle PL/SQL is a procedural language that is similar to SQL. It consists of subprograms which are executed in the database. Stored procedures and stored functions are both Oracle PL/SQL subprograms that are able to be stored and executed. The main difference between functions and procedures is that functions must return a value, while stored procedures do not need to return a value.

The syntax for creating stored procedures and functions is nearly the same. They both follow the basic PL/SQL block structure, which consists of an optional declarative section for variables and constants, a mandatory executable part starting with the keyword “BEGIN” and ending with the keyword “END”, and then an optional exception handling section. (Oracle, 2015)

The general syntax for creating a stored procedure is documented on Oracle as follows:

```
CREATE OR REPLACE procedure procedure_name(arg1 data_type, ...) AS
BEGIN
  ....
END procedure_name;
```

The general syntax for creating a stored function is similar to that of creating a stored procedure. It is documented on Oracle as follows:

```
CREATE OR REPLACE function procedure_name(arg1 data_type, ...) AS
BEGIN
  ....
END procedure_name;
```

Subprograms have several benefits. First of all, they extend SQL past its standard use. Subprograms have the capability of using variables, constants, program control flow, and cursors. The modularity and reusability is also a benefit. Procedures and functions can perform operations based on input parameters. (Oracle, 2015, Advantages of Subprograms)

4.1.2 Packages

Stored procedures and functions may be created to standalone, or they may be created as an addition to a package. A package is a grouping to help formally state the relationships of its contents. It consists of two main sections. First, there is a specification section for the package itself which contains variables global to the package and subprogram declarations. The second part is a body section of the implementation of the stored procedures and functions. It is better to group stored procedures and functions within a package rather than have them stand alone for several reasons. First, stored procedures and functions grouped within a package have access to more complex variable types when they are contained within a package. These include cursors to iterate results, and result set variable types themselves. In addition, packages are more portable for when the database needs to be moved.

4.1.3 Triggers

Triggers are SQL statements that implicitly execute stored procedures when a specified event within a given database occurs. The source of the event does not affect whether or not the trigger is fired; only disabling the trigger will cause it to not fire. The purpose of triggers is to enforce rules that cannot otherwise be enforced through constraints. Some examples of such rules from Oracle documentation: (Oracle, Triggers, 2015)

- Automatically generate derived column values
- Prevent invalid transactions
- Enforce complex security authorizations
- Enforce referential integrity across nodes in a distributed database
- Enforce complex business rules
- Provide transparent event logging
- Provide auditing
- Maintain synchronous table replicates
- Gather statistics on table access
- Modify table data when DML statements are issued against views
- Publish information about database events, user events, and SQL statements to subscribing applications

More precisely however, triggers can fire procedures on commands that alter the entries within tables: INSERT, DELETE, and UPDATE (DML statements). They can also act on modifications to the database schema or overall instance as well (DDL statements), usually ALTER and CREATE. Finally, they can be issued on specific database events, such as logon/logoff, errors, startup/shutdown, or explicit calls by a schema/user.

To fully define a trigger, several things are needed. The point in time the trigger executes relative to the triggering statement must be specified. There are 4 possible options for when to execute:

BEFORE
Executes trigger's procedure before the triggering statement runs.

AFTER
Executes trigger's procedure after the triggering statement has completely finished.

BEFORE … FOR EACH ROW
Executes tigger's procedure before the triggering statement is run for each specific row it acts on.

AFTER … FOR EACH ROW
Executes trigger's procedure after each run of the triggering statement on a row.

The command the trigger executes on, the table being watched, and the condition for execution must also be specified. Triggers can also be defined as INSTEAD OF, which indicates the trigger should replace the triggering statement, rather act with it. The syntax for creating a trigger follows: (PL/SQL Tutorials, 2015)

```
CREATE OR REPLACE TRIGGER <trigger name>
<TIME> <COMMAND> [OF <attribute list>] ON <table>
[REFERENCING <attribute> AS <new attribute name>]
[FOR EACH ROW]
WHEN (condition)
[DECLARE local variables]
BEGIN
<PL/SQL statements to execute>
END;
  /
```

where, anything inside [ ] is optional, and:
TIME may be AFTER, BEFORE, or INSTEAD OF

COMMAND may be UPDATE, INSERT, DELETE, ALTER, CREATE, etc.

Sample trigger:

```
CREATE OR REPLACE TRIGGER Print_salary_changes
  BEFORE DELETE OR INSERT OR UPDATE ON Emp_tab
  REFERENCING Empno AS eid
  FOR EACH ROW
WHEN (new.eid > 0)
DECLARE
    sal_diff number;
BEGIN
    sal_diff  := :new.sal  - :old.sal;
    dbms_output.put('Old salary: ' || :old.sal);
    dbms_output.put('  New salary: ' || :new.sal);
    dbms_output.put_line('  Difference ' || sal_diff);
END;
/
```

Section 4.2 Example Subprograms

This section describes a few subprograms for this database. It describes the subprograms, shows the subprograms themselves, and displays their output.

For this database, there are **two stored procedures** and **one stored function** contained within **a package**. The first stored procedure inserts a new record into the holiday table. Its parameters take the day and month as a string, the name of the holiday, and the holiday type indicating whether it's a full or half day off. The second stored procedure deletes an ingredient from being used in a menu item. It accepts the ingredient id as a parameter. The function takes seven date strings as a parameter in order to get the schedule for that week. It expects the days in order of Monday to Sunday. It returns a pipelined record result.

Next, there are **three triggers** defined. The first handles a delete operation on a view. Normally, Oracle will not allow delete operations on views, but triggers can be written to catch the delete and operate accordingly on the base tables. The view being deleted from is a list of all employees who are training for a position. The trigger runs instead of deleting on the view. It changes the training status in the base table has_position, so the view reflects the changes made to the base table. The second trigger is a before update trigger. It handles an attempt to update a primary key on a transaction by changing all of the corresponding foreign keys in the sold_in table first. The third trigger is a before delete trigger. It handles an attempt to delete an employee before corresponding foreign keys are removed, by going through and deleting all of the corresponding foreign keys before the primary key is deleted.

Package with stored procedures and functions

```
/*
 * Package Specification Section
 */
create or replace package bajs_pkg as

-- Procedure Prototypes
procedure add_holiday( mmdd varchar2, name varchar2, t varchar2);
procedure delete_ingredient(iid number);

-- Type Definitions
type sch_record is table of bajs_schedule%rowtype;

-- Function Prototypes
function get_schedule(
    mon varchar2,
    tue varchar2,
    wed varchar2,
    thu varchar2,
    fri varchar2,
```

```
    sat varchar2,
    sun varchar2
) return sch_record pipelined;

end bajs_pkg;
/


/*
 * Package Body Section
 */
create or replace package body bajs_pkg as

    -- Delete an Ingredent from being used in a menu item by supplying the ingredient id
    procedure delete_ingredient (iid number) is
    begin
        delete from bajs_used_in u
        where u.ingredient_id = iid;
    end delete_ingredient;

    -- Create a new Holiday Record
    procedure add_holiday( mmdd varchar2, name varchar2, t varchar2) is
    begin
        -- TODO give the type a default, and
        insert into BAJS_HOLIDAY values( mmdd, name, t);
    end add_holiday;

    -- input date strings as 'dd/mm/yyyy' for each corresponding weekday
    -- function expects the correct weekdays in the order of monday to sunday
    -- returns columns (sid, tname, we_st, we_ed, wd_st, wd_ed, mon, tue, wed, thu,
    --                  fri, sat, sun, location)
    function get_schedule(
        mon varchar2,
        tue varchar2,
        wed varchar2,
        thu varchar2,
        fri varchar2,
        sat varchar2,
        sun varchar2
    ) return sch_record
    pipelined as

    -- Define Cursor
    cursor temp_cur is
    select * from
    (
        select * from bajs_sch_template
        natural join
        (
            select * from (
              -- Monday
                select sid, fname as mon
                from bajs_sch_hist
                where day = to_date(mon, 'dd/mm/yyyy')
            )
            natural join
            ( -- Tuesday
                select sid, fname as tue
                from bajs_sch_hist
                where day = to_date(tue, 'dd/mm/yyyy')
            )
            natural join
            ( -- Wednesday
                select sid, fname as wed
                from bajs_sch_hist
                where day = to_date(wed, 'dd/mm/yyyy')
            )
            natural join
            ( -- Thursday
                select sid, fname as thu
                from bajs_sch_hist
                where day = to_date(thu, 'dd/mm/yyyy')
```

```
            )
            natural join
            ( -- Friday
                select sid, fname as fri
                from bajs_sch_hist
                where day = to_date(fri, 'dd/mm/yyyy')
            )
            natural join
            ( -- Saturday
                select sid, fname as sat
                from bajs_sch_hist
                where day = to_date(sat, 'dd/mm/yyyy')
            )
            natural join
            ( -- Sunday
                select sid, fname as sun
                from bajs_sch_hist
                where day = to_date(sun, 'dd/mm/yyyy')
            )
        )
    ); -- End Cursor Definition
    begin

        -- Iterate Cursor to return rows
        for cur_rec in temp_cur loop
        pipe row(cur_rec);
        end loop;

    end get_schedule;
end bajs_pkg;
/
```

Execute Function get_Schedule()

```
/*
 * Call function get_schedule(<params>) from Package bajs_pkg
 */
select * from table(bajs_pkg.get_schedule(
         '02/11/2015',
         '03/11/2015',
         '04/11/2015',
         '05/11/2015',
         '06/11/2015',
         '07/11/2015',
         '08/11/2015'));
```

ID	TNAME	WE_ST	WE_ED	WD_ST	WD_ED	LOCATION	MON	TUE	WED	THU	FRI	SAT	SUN
1	Opening Sup/Mgr	600	1400	700	1500	front	John	John	Susan	Bob	John	Bob	Bob
2	Support Sup/Staff	700	1500	700	1500	front	Jane	Jane	Chris	Susan	Chris	Susan	Susan
3	Front (tomatoes)	800	1500	800	1500	front	Maria	Deniz	Li	Deniz	Jin	Deniz	Deniz
4	Front (dressings)	800	1530	800	1530	front	Deniz	Jules	Maria	Jules	Udo	Maria	Maria
5	Front (register 1)	1100	1600	1100	1600	front	Jules	Sammie	Deniz	Sammie	Chike	Udo	Udo
6	Front	900	1400	900	1400	front	Li	Hadley	Udo	Hadley	Hadley	Jin	Jin
7	Busser	1115	1515	1115	1515	front	Earl	Earl	Earl	Ash	Ash	Ash	Ash
8	Cheese	600	1430	700	1500	front	Sal	Taonga	Mavuto	Steph	Steph	Taonga	Taonga
9	Meat	600	1430	700	1500	front	Taonga	Aston	Sal	Mavuto	Mavuto	Sal	Sal
10	Closing Supervisor	1200	2030	1200	2030	front	Bob	Bob	John	Chris	Susan	Chris	Chris
11	Cold Prep	1300	2030	1300	2030	front	Aston	Sal	Taonga	Aston	Sal	Mavuto	Mavuto
12	Register 1	1530	2030	1530	2030	front	Sammie	Chike	Jin	Chike	Sammie	Hadley	Hadley
13	Expedite	1330	2030	1330	2030	front	Hadley	Udo	Chike	Udo	Li	Sammie	Sammie
14	Front/Busser	1600	2000	1600	2000	front	Chike	Jin	Hadley	Jin	Jules	Chike	Chike
15	Kitchen Sup or Staff	700	1500	700	1500	kitchen	Kamala	Kamala	Kamala	Kamala	Lei	Lei	Lei
16	Bakery	700	1500	700	1500	kitchen	Zephyrus	Zephyrus	Shyama	Shyama	Ulyses	Paora	Paora
17	Grill	700	1500	700	1500	kitchen	Ade	Marcelli	Egill	Kasey	Ade	Gina	Gina
18	Salads	700	1500	700	1500	kitchen	Gina	Gina	Aminta	Nikolas	Devraj	Aminta	Aminta
19	Kitchen Sup or Staff	1330	2030	1330	2030	kitchen	Kasey	Aminta	Kasey	Gina	Gina	Kasey	Kasey
20	Grill	1630	2030	1630	2030	kitchen	Egill	Lei	Ade	Aminta	Aminta	Ade	Ade
21	Salads	1500	2030	1500	2030	kitchen	Devraj	Devraj	Devraj	Marcelli	Marcelli	Marcelli	Marcelli
22	Support Staff	1630	2030	1630	2030	kitchen	Vinzent	Gerhard	Gerhard	Devraj	Gerhard	Gerhard	Gerhard
23	Closing Janitor	1500	2030	1500	2030	kitchen	Mavuto	Lenny	Borya	Borya	Lenny	Borya	Borya
24	Opening Janitor	730	1500	730	1500	kitchen	Borya	Steph	Steph	Lenny	Mavuto	Lenny	Lenny
25	Support Staff	700	1500	700	1500	kitchen	Gerhard	Vinzent	Vinzent	Vinzent	Vinzent	Vinzent	Vinzent

```
25 rows selected.
```

Execute Procedure delete_ingredient()

```
CS342 SQL> select * from bajs_used_in
  2  where ingredient_id = 5;

MENU_ITEM_ID INGREDIENT_ID   QUANTITY
------------ ------------- ----------
           5             5          5
           6             5          5
           8             5          5

CS342 SQL> exec bajs_pkg.delete_ingredient(5);

PL/SQL procedure successfully completed.

CS342 SQL> select * from bajs_used_in
  2  where ingredient_id = 5;

no rows selected
```

Execute Procedure add_holiday()

```
CS342 SQL> select * from bajs_holiday;

HDATE      NAME                TYPE
---------- ------------------- -----
12/25      Christmas           full
12/24      Christmas Eve       half
12/31      New Years Eve       half
01/01      New Years Day       half
10/31      Halloween           half
11/27      Thanksgiving        half
06/07      Easter              full

7 rows selected.

CS342 SQL> exec bajs_pkg.add_holiday('11/12', 'Veterans Day', 'half');

PL/SQL procedure successfully completed.

CS342 SQL>
```

Triggers

```
/*
    Instead of deleting a training employee from the view, remove their training
    status from the base table has_position.
*/
create or replace trigger bajs_update_train_emp
instead of delete on bajs_training_emp
for each row
begin
    update bajs_has_position p
    set p.is_training =  0
    where p.employee_id = :old.eid and p.position_id = :old.pid;
end;
/
commit;

/*
    Make sure foreign key is updated before primary key
*/
```

```
create or replace trigger bajs_change_tid
before update on bajs_transaction
for each row
when (old.id != new.id)
begin
    update bajs_sold_in set transaction_id = :new.id
    where :old.id = transaction_id;
end;
/
commit;
/*
    Make sure employee_id foreign key is removed before removing employee
*/
create or replace trigger bajs_emp_delete
before delete on bajs_employee
for each row
when (old.id > 0)
begin
    delete from bajs_user u
    where :old.id = u.employee_id;

    delete from bajs_requests_vacation r
    where :old.id = r.requested_by;

    update bajs_requests_vacation r set r.responded_by = null
    where :old.id = r.responded_by;

    delete from bajs_orders o
    where :old.id = o.employee_id;

    delete from bajs_is_scheduled_for i
    where :old.id = i.employee_id;

    delete from bajs_has_position h
    where :old.id = h.employee_id;

    delete from bajs_employment_history eh
    where :old.id = eh.employee_id;

    delete from bajs_cannot_work_with c
    where :old.id = c.employee1_id or :old.id = c.employee2_id;

    delete from bajs_availability a
    where :old.id = a.employee_id;
end;
/
commit;
```

Example Update Training Employee Trigger

```
CS342 SQL> select * from bajs_training_emp
  2  order by eid;

       EID       PID TITLE               NAME
---------- ---------- -------------------- ----------
         3          1 Supervisor           Bob
         6          7 Salads               Kamala
         6          4 Kitchen Supervisor   Kamala
         6          5 Bakery               Kamala
         9          3 Cold Prep            Taonga
        14          2 Cashier              Deniz
        23          9 Janitor              Lenny
        33          7 Salads               Aminta
        36          8 Other Kitchen        Gerhard
        39         10 Busser               Ash

10 rows selected.

CS342 SQL> delete from bajs_training_emp where eid = 6 and pid = 4;
```

```
1 row deleted.

CS342 SQL> select * from bajs_training_emp
  2  order by eid;

       EID       PID TITLE               NAME
---------- ---------- ------------------- ----------
         3          1 Supervisor          Bob
         6          7 Salads              Kamala
         6          5 Bakery              Kamala
         9          3 Cold Prep           Taonga
        14          2 Cashier             Deniz
        23          9 Janitor             Lenny
        33          7 Salads              Aminta
        36          8 Other Kitchen       Gerhard
        39         10 Busser              Ash

9 rows selected.
```

Example Change Transaction Id Trigger

```
CS342 SQL> select * from bajs_transaction
  2  where id = 9;

       ID ON_DATE
---------- ---------
        9 02-DEC-14

CS342 SQL> update bajs_transaction
  2  set id = 9999
  3  where id = 9;

1 row updated.

CS342 SQL> select * from bajs_transaction
  2  where id = 9;

no rows selected

CS342 SQL> select * from bajs_transaction
  2  where id = 9999;

       ID ON_DATE
---------- ---------
      9999 02-DEC-14

CS342 SQL>
```

Example Delete Employee Trigger

```
/*
    Test emp_delete trigger
*/
select * from bajs_employee where id=12;
select * from bajs_user u where employee_id=12;
select * from bajs_requests_vacation r where requested_by=12 or responded_by=12;
select * from bajs_orders o where employee_id=12;
select * from bajs_is_scheduled_for i where employee_id=12;
select * from bajs_has_position h where employee_id=12;
select * from bajs_employment_history eh where employee_id=12;
select * from bajs_cannot_work_with c where employee1_id=12 or employee2_id=12;
```

```
select * from bajs_availability a where employee_id=12;

delete from bajs_employee where id=12;

select * from bajs_employee where id=12;
select * from bajs_user u where employee_id=12;
select * from bajs_requests_vacation r where requested_by=12 or responded_by=12;
select * from bajs_orders o where employee_id=12;
select * from bajs_is_scheduled_for i where employee_id=12;
select * from bajs_has_position h where employee_id=12;
select * from bajs_employment_history eh where employee_id=12;
select * from bajs_cannot_work_with c where employee1_id=12 or employee2_id=12;
select * from bajs_availability a where employee_id=12;
```

```
        ID FIRST_NAME LAST_NAME  IS_MANAGER BIRTH_DAT MAX_HRS_WEEK PHONE_NUMB CLOCK_NUMBER
---------- ---------- ---------- ---------- --------- ------------ ---------- ------------
        12 Steph      Vivian             0 04-DEC-88          40 1234567890          12


EMPLOYEE_ID EMAIL             PASSWORD       API_TOKEN
----------- ----------------------------- ------------------------------
12          emp12@example.com password       lsdn4%og23blsfqa~>8f&;aeri


ID REQUESTED_BY RESPONDED_BY IS_APPROVED START_DATE_TIME END_DATE_TIME
-------------------------------------------------------------------------------
12           12          10           0  13-SEP-02 02.10.10.123000 PM 13-SEP-02 02.20.10.123000 PM
32           12          10           0  23-OCT-02 02.10.10.123000 PM 24-OCT-02 02.10.10.123000 PM
52           12           4           0  22-NOV-02 02.10.10.123000 PM 23-NOV-02 02.10.10.123000 PM


no rows selected


EMPLOYEE_ID   SHIFT_ID ON_DATE
----------- ---------- ---------
         12          8 05-NOV-15
         12          8 06-NOV-15
         12         24 03-NOV-15
         12         24 04-NOV-15


EMPLOYEE_ID POSITION_ID DATE_ACQU DATE_REMO IS_PRIMARY IS_TRAINING
----------- ----------- --------- --------- ---------- -----------
         12           3 04-DEC-11                 1           0
         12           9 04-DEC-11                 1           0


EMPLOYEE_ID DATE_EMPL DATE_UNEM
----------- --------- ---------
         12 24-DEC-09


EMPLOYEE1_ID EMPLOYEE2_ID
------------ ------------
          10           12


EMPLOYEE_ID DAY        STARTT       ENDT
----------- -------- ---------- ----------
         12 mon            600       2100
```

```
        12 tue            600       2100
        12 wed            600       2100
        12 thu            600       2100
        12 fri            600       2100
        12 sat            600       2100
        12 sun            600       2100

7 rows selected.

1 row deleted.

no rows selected

no rows selected


no rows selected

no rows selected

no rows selected

no rows selected

no rows selected

no rows selected

no rows selected

no rows selected

Commit complete.
```

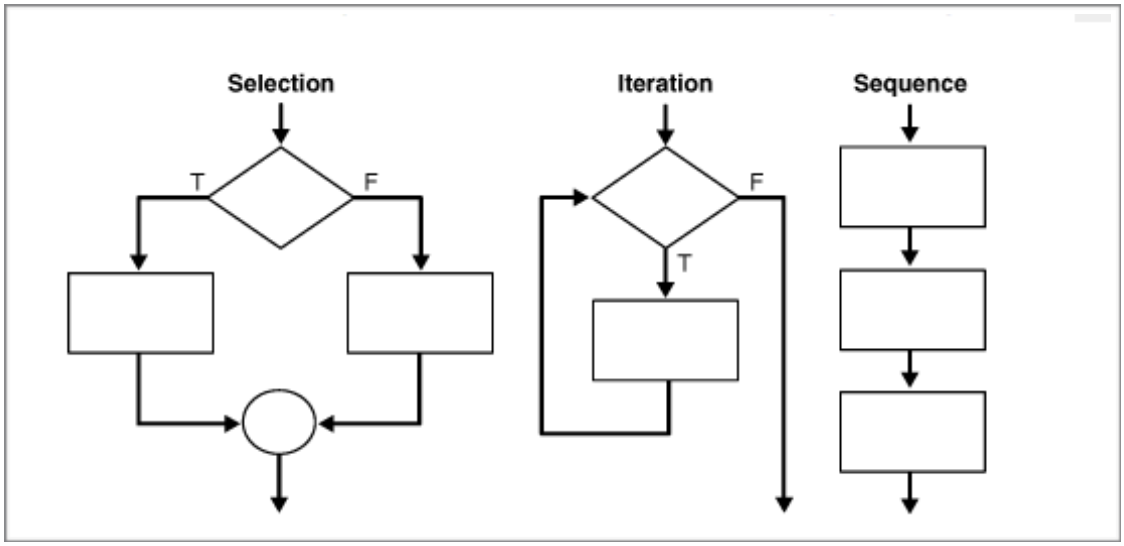
Section 4.3 PL/SQL Language

This section describes functionality provided by the Oracle PL/SQL Language. It describes the available types for variables and constants, the possible program control flow and composite data structures.

4.3.1 Variables and Constants

Oracle PL/SQL allows for user defined variables and constants. There are several different types of variables that can be defined. There are regular SQL types of variables such as VARCHAR2, DATE and NUMBER. There are data types that may only be used by PL/SQL which are BOOLEAN, RECORD, REF CURSOR, INDEX BY TABLE, and PLS_INTEGER. Constants will be defined using the CONSTANT clause. There are convenient ways to assign variables to any corresponding column types. The first, %ROWTYPE, lets a composite variable take on the same values and names from a row of a specific table. The second, %TYPE, lets a variable take on the value of a single column from a specific table and column in that table. Assignments are made to variables with the := operator.
(Oracle, Using Variables and Constants, 2015)

4.3.2 Control Program Flow



(Oracle, Controlling Program Flow, 2015)

Oracle PL/SQL allows for control of program flow. There are three general methods of controlling program flow called selection, iteration and sequence. Selection is a conditional statement that

evaluates to true or false, and directs the program accordingly. A basic selection is an IF statement with optional following ELSE IF and ELSE statements. Another selection is a CASE … WHEN which is similar to a switch statement used on other programming languages. The second type of program control flow is iteration. Iteration includes loops through data including a FOR … LOOP, WHILE … LOOP, and LOOP .. EXIT WHEN. Iterative statements are used for composite data structures such as record variable types where there are several rows that need to be traversed. Finally, sequence is the simplest program control flow, which is the sequential evaluation of statements (Oracle, Controlling Program Flow, 2015).

4.3.3 Composite Data Structures and Cursors

Oracle PL/SQL allows for more complex objects called composite data structures. A composite data structure is a record that is either setup as an existing table, or a user defined structure like a table. It consists of a group of related items that each have their own name and type.

Operations on records require traversing the data row by row. This is done by the use of a pointer called a cursor. There are two types of cursors; implicit and explicit. An implicit cursor does not require code to process the cursor itself and has less options available than an explicit cursor. An explicit cursor requires code to set it up, but allows for more options for processing data with it which are attributes that give details about the data. These include %NOTFOUND, %FOUND, %ROWCOUNT, and %ISOPEN. (Oracle, 2015, Retrieving Data from a Set Using Cursors and Cursor Variables)

A simple example of a cursor is listed on Oracle as follows: (Oracle, 2015, Retrieving Data from a Set Using Cursors and Cursor Variables)

```
DECLARE
  CURSOR cursor_name type IS query_definition;
OPEN cursor_name
  LOOP
    FETCH record;
    EXIT WHEN cursor_name%NOTFOUND;
    ...;             -- process fetched row
  END LOOP;
CLOSE cursor_name;
```

The cursor is declared with a query that is the result set it will iterate through. Next, The cursor is opened with the keyword OPEN, and then a loop is set up so the cursor can FETCH each record in

the result set. Operations on the result set are done inside the loop, and then the loop ends. Finally, the cursor is closed with CLOSE when it is no longer needed.

Section 4.4 PL/SQL Like Tools (Oracle, MS SQL Server and MySQL)

Although specifications have been developed to standardize the language used by every DBMS as SQL, several versions of SQL exist due to specializations made by major DBMS developers. Some of these languages are covered in this section.

PL/SQL

This language is specific to Oracle databases, and has already been discussed above. There are other PL/SQL like tools available for different databases. MS SQL Server uses Transact-SQL, and MySQL has the language extensions built in.

-Sample PL/SQL procedure syntax:

```
CREATE OR REPLACE procedure procedure_name(arg1 data_type, ...) AS
BEGIN
  ....
END procedure_name;
```

Transact-SQL

This is a proprietary language developed and used by Microsoft and Sybase. Much like PL/SQL the language extends SQL standardization by adding features for procedural programming, local variables, triggers, and various support functions like mathematics, data processing. In addition, changes are made to the normal syntax of some SQL commands, such as DELETE and INSERT. (Transact-SQL, Wikipedia, 2015)

-Sample Transact-SQL procedure syntax:

```
DELIMITER //
CREATE PROCEDURE GetAllProducts()
   BEGIN
   SELECT *  FROM products;
   END //
DELIMITER ;
```

MySQL

This is an open source language, under the GNU Licence, originally developed and sponsored by a for-profit Swedish firm, MySQL AB. It is now owned and maintained by Oracle Corporation, and is one

of the most popular DBMS languages in the world. Two versions are offered, the open source MySQL Community Server, and the proprietary Enterprise Server. Like the above two SQL languages, MySQL extends upon SQL by implementing procedural programming, local variables, triggers, and various support functions like cursors, sub-select statements, and new types of schema. Some notable limitations to the language is the inability to define triggers on views, and triggers being limited to one action per timing. (MySQL, Wikipedia, 2015)

-Sample MySQL procedure syntax:

```
mysql> delimiter //

mysql> CREATE PROCEDURE simpleproc (OUT param1 INT)
    -> BEGIN
    ->   SELECT COUNT(*) INTO param1 FROM t;
    -> END//
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;
CALL simpleproc(@a);
```

Phase 5 Graphical User Interface Design and Implementation

Section 5.1 Daily Activities of User Groups

This section describes the permissions and functionality available for each user group that may access the front end application. It also describes each user group's expected daily activities while accessing the front end

5.1.1 Daily Activities of User Groups

Two user groups are available for accessing the front end website: employees and managers. Managers will need to make heavier use of the front end on a daily basis due to their daily responsibilities, and correspondingly are provided more functionality and access available to them.

Once any user has logged in, the site's home page shows weekly schedules that have been finalized by managers. The focus of this website is to provide a centralized point for modifying and viewing work schedules, and employees are only expected to check the schedule on a daily basis to determine their hours for the week, since they will not be modifying the schedules. The employee user group is therefore only given access to the site's home page.

Managers on the other hand need to be able to create and modify work schedules, in addition to viewing the schedule, and are therefore given access to additional web pages. Upon login, managers are taken to the home page, like employees, where they may view schedules.

In addition, however, managers may visit the "Employee" tab where the can search for any employee, active or inactive. This tab allows managers to view and modify basic information about the employee such as the employee's name and birthday, as well as scheduling information such as weekly availabilities as positions worked by the employee. The employee's employment history with the company is also listed on this tab. Unlike the other modifiable pieces of information on this tab, employee availability is likely to change by the week, so it is expected managers will be visiting this tab often to adjust this information in preparation for creating next week's schedule.

The "Scheduling" tab is also available to managers, where they may adjust which employees are scheduled for the shifts in a given week. It is expected managers will visit this page the most while accessing the site, most likely on a daily basis, in order to create and prepare the following week's

schedule. Correspondingly, most of the website's functionality is focused on this specific webpage, in order a facilitate quickly and easily scheduling employees for shifts. For example, to quickly populate an upcoming week's schedule, managers may import the schedule from the previous week if they know the following week's schedule is going to be very similar. This reduces the amount of shifts that need to explicitly scheduled down to the few that need to be changed. Further features on this page are described below while discussing the major features on the GUI.

Finally, a "Data" tab is provided for managers to access. This page checks data on the currently active schedule to calculate the totals number days and hours each employee works on that given week. The information is presented in graphical format using charts and graphs, allowing managers to easily visualize how much work employees are being given in comparison to each other. Managers are expected to visit this page on a weekly basis before finalizing the following week's schedule in order to compare how much work employees are being given. As development continues, it is expected some of these graphs will be included in the employee tab in a modified form so that managers may visualize this information on a per employee basis. These graphs will also continue to be developed in order to cover the aggregate data presented over wider time ranges.

5.1.2 Itemized Description of User Groups

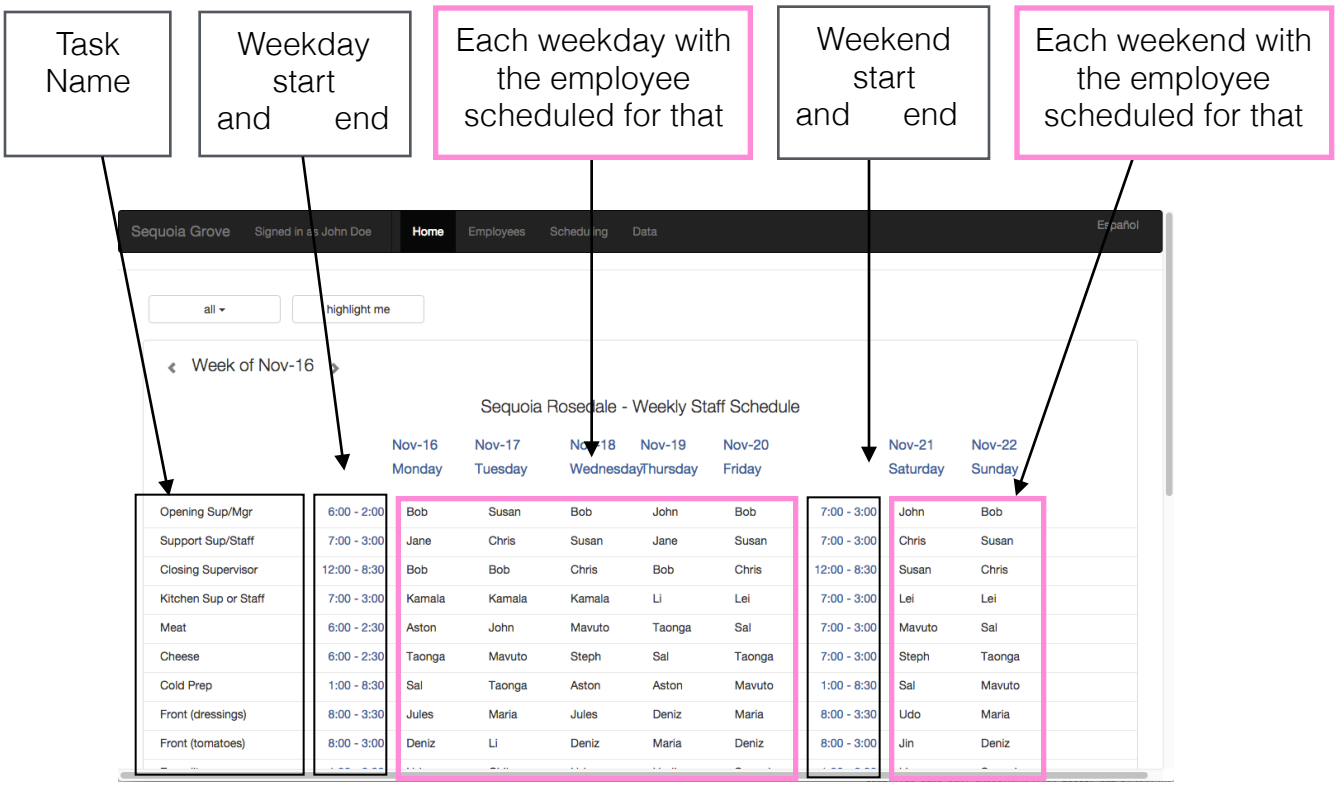
Employee Group:
- View Weekly Schedules

Manager Group:
- View Weekly Schedules
- Edit/Create Weekly Schedules
- View/Edit Employee Information:
- Basic Information:
  - Name
  - Birthday
  - Phone Number
  - Clock Number
  - Manager Status
- Weekly Availabilities
- Workable Positions
- Employment History (viewable only)

View Aggregate Data: Hours/Days of Schedule Employees for Week

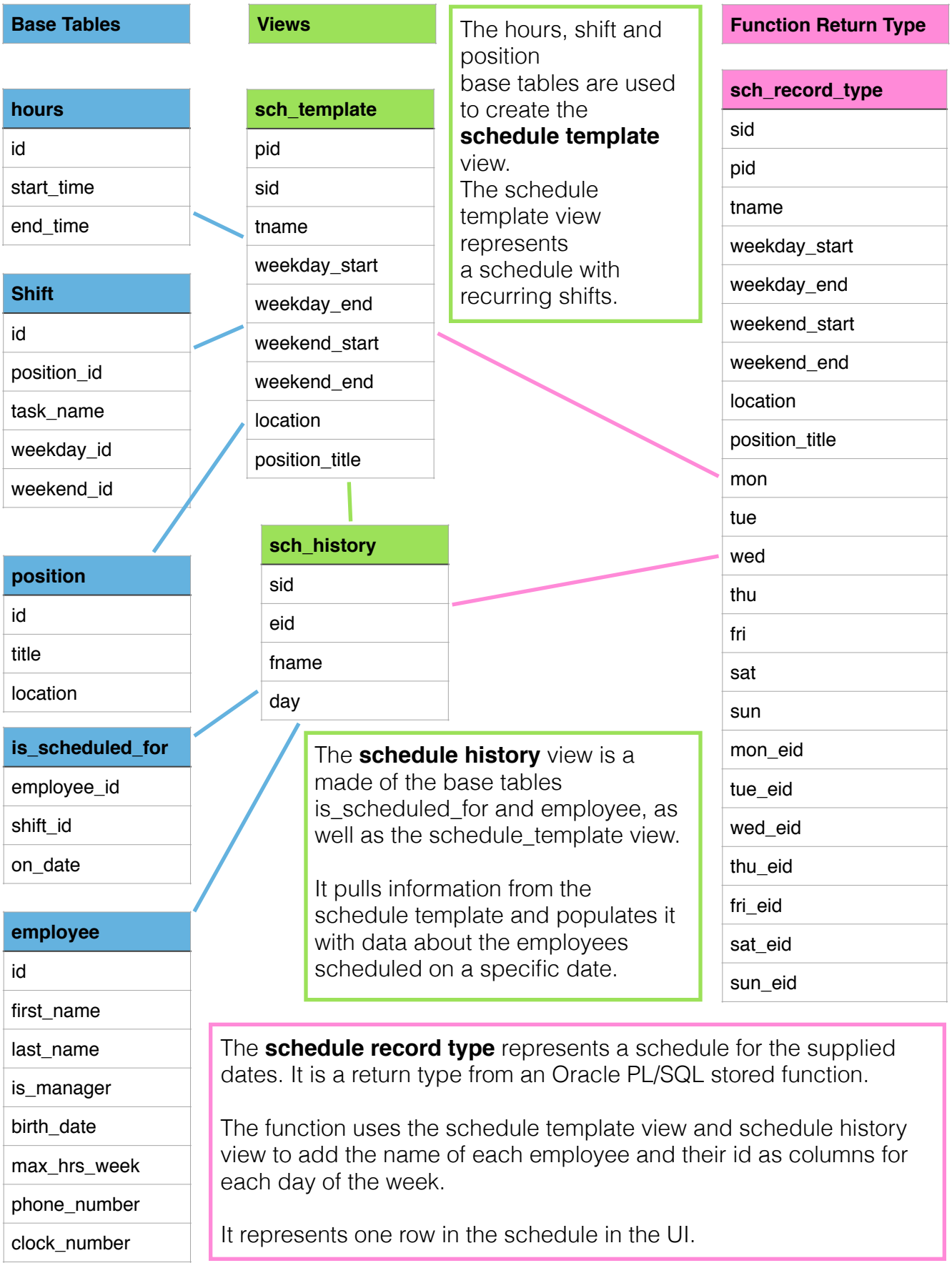
Section 5.2 Relations Views and Subprograms Related to Activities

The **Schedule View UI** displays the schedule. For any given row on a schedule, the following information must be collected. The task name, and shift hours can be directly collected from base tables and views, however, the information regarding specific employees scheduled for a given shift on a specific date needed a function that would accept parameters in order to match



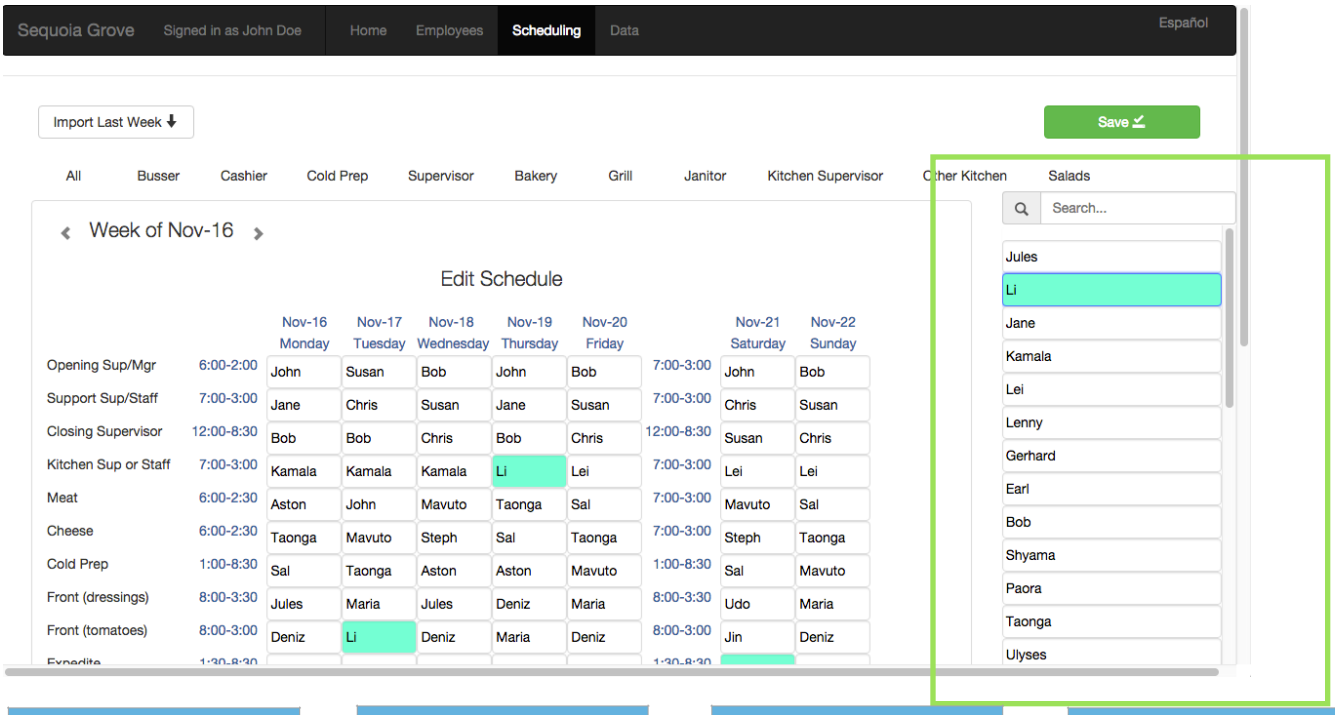
The UI for the schedule view was constructed of base tables, views and a stored function as shown on the next page.

Diagram of the base tables, views and the function used to construct the schedule view.



The **Schedule Edit UI** is built the same as the Schedule View UI, however, it has built in functionality to update the schedule. The front end keeps a list of any changes made to the schedule, and inserts them into the database. In order to avoid duplicate employees scheduled on the same day, a stored procedure was created to implement the update. The stored procedure will check if the given shift and date already exist in the database. If the combination of the scheduled date and shift already exist, it will update the employee to the new one. If the scheduled date and shift do not exist, it will create a new row in is_scheduled_for with the supplied employee id, shift id, and date.

The schedule edit UI uses a view to construct a list of standard employees that may be put on the schedule. The view that this list reads from has only employees that are currently employed and has their current positions that they work associated with them. This view is convenient for sorting the employees by their position, and making sure that employees are current employees.



Base Tables

employee

- id
- first_name
- last_name
- is_manager
- birth_date
- max_hrs_week
- phone_number
- clock_number

has_position

- employee_id
- position_id
- date_acquired
- date_removed
- is_primary
- is_training

position

- id
- title
- location

employment_history

- employeee_id
- date_employed
- date_unemployed

View

std_employee

- eid
- name
- pid
- position_title

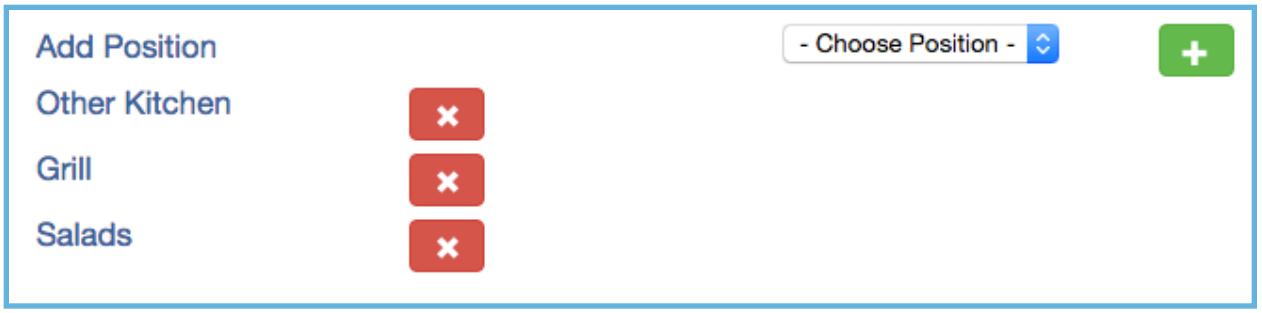
The **standard employee** view is a view that shows employees that are current, and their current positions. This means that the employees do not have a date_unemployed set in their employment history, and they do not have a date_removed set in the has_position table. The employee and position table are used to associate other information with the view.

Section 5.3 Screenshots and Usability

The **Schedule View** is made to be convenient for the user. The schedule can be narrowed down by three different categories of shifts: specific to the current logged in user, specific to the location, or specific to the type of position. The filtering is convenient because different positions have groups of shifts, and different locations have groups of positions. In addition, the user can click the “Highlight Me” button, so it will dynamically apply a css class to highlight all occurrences of that user in the schedule by matching each day and shift that contains that employee’s id.

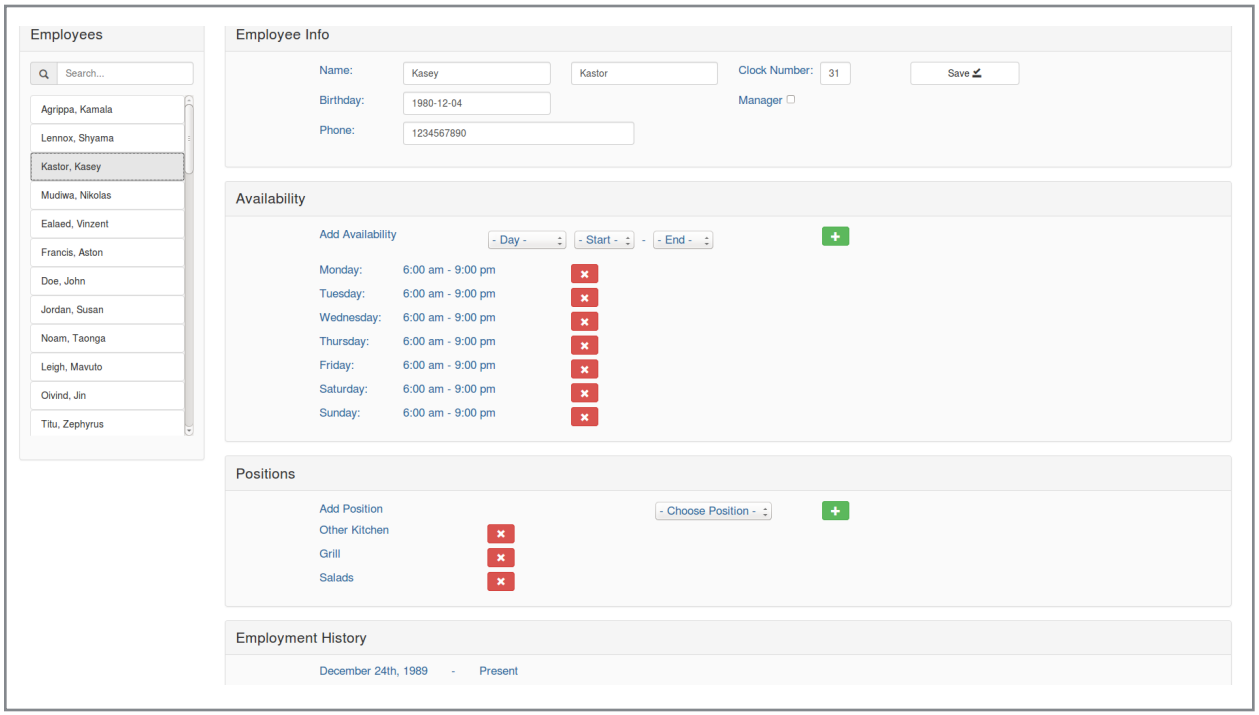
The schedule view defaults to displaying the current week. Upon initialization of the application, it gets the current date and

Positions - add positions by selecting the position title, and clicking the green + button. remove positions by clicking the corresponding red X button.



Employment History - displayed for informative purposes

The **Employee Edit UI** uses several tables to gather information on employees. This information is then able to be edited by the user. New information added is saved in the database, and information removed is marked with a date_delete property, or removed from the database. There was a trigger that utilized a sequence in order to keep employee ids unique without having to manually determine the next id to insert.



Base Tables

employee
id
first_name
last_name
is_manager
birth_date
max_hrs_week
phone_number
clock_number

availability
employee_id
day
startt
endt

employment_history
employeee_id
date_employed
date_unemployed

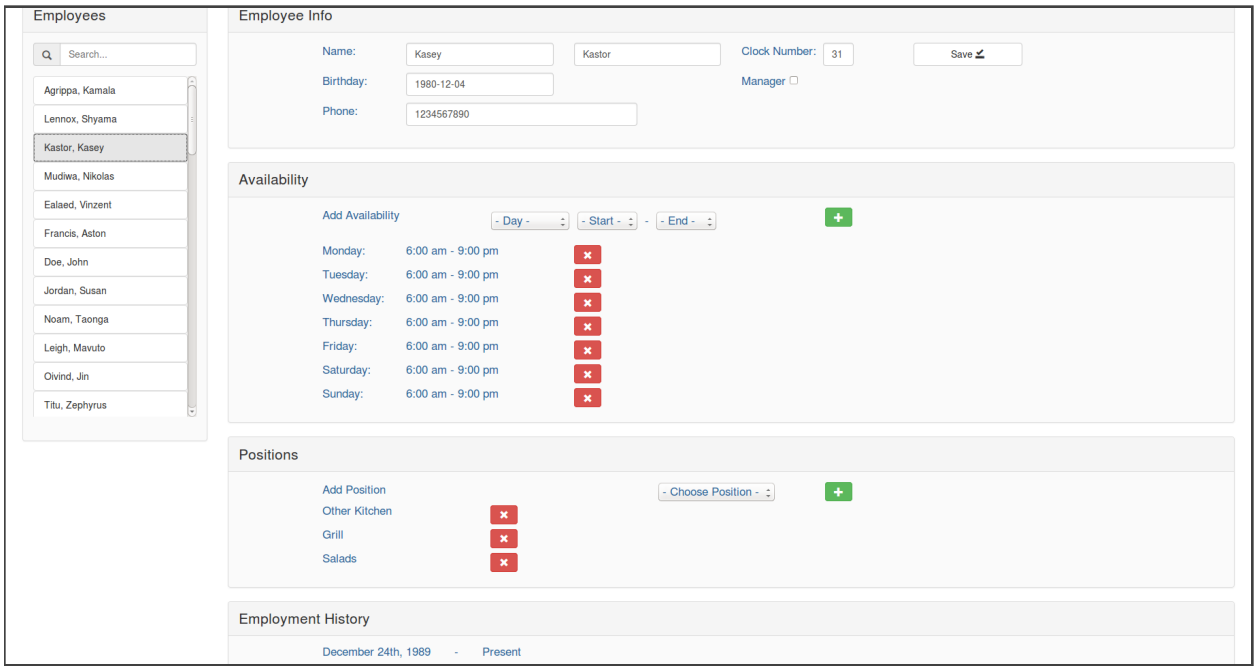
emp_all_info
employee_id
first_name
last_name
is_manager
birth_date
max_hrs_week
phone_number
clock_number
day
startt
endt
date_employed
date_unemployed
position_id
title
location

position
id
title
location

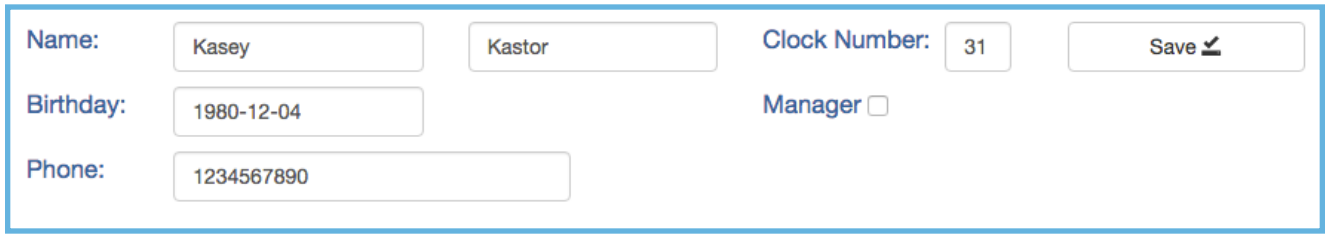
has_position
employee_id
position_id
date_acquired
date_removed
is_primary
is_training

The **employee all info** view is a view that collects all of the relevant information about each employee that can be updated in the front end. The employee edit UI uses both the base tables and employee all info view to pull information about each employee.

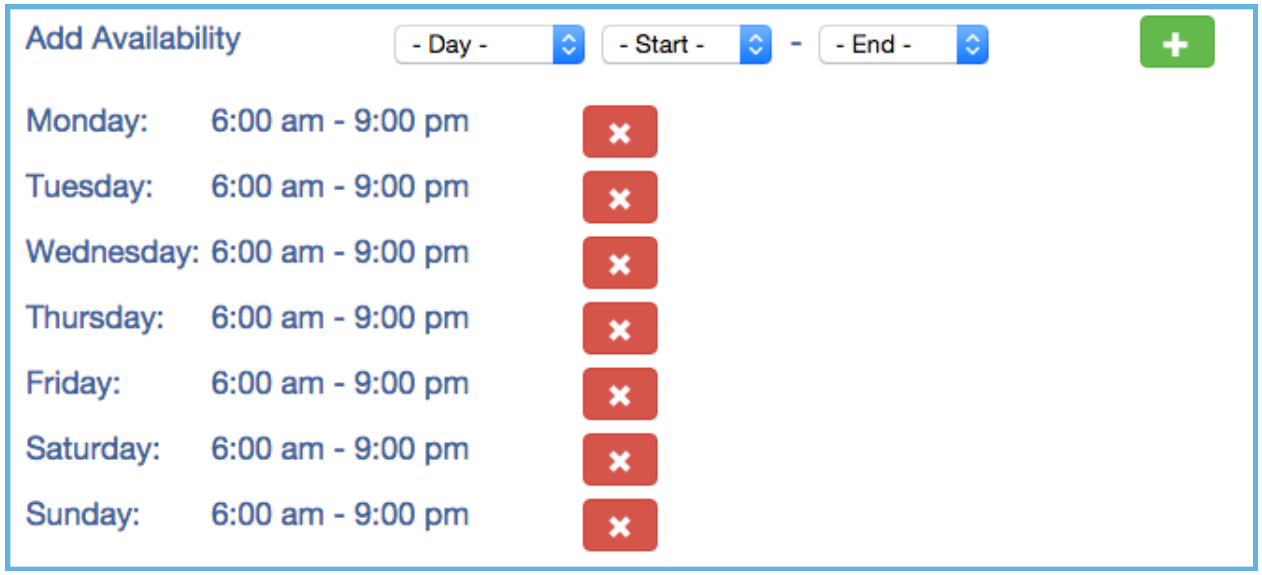
The **Employee View** allows the user to update current employees or add new employees. The list of employees is similar to the list displayed on the schedule edit view, however, it contains all employees (not just current ones), and it also contains more information about each employee. The employee's basic information can be updated, availability times can be added or removed and positions they can work can be added or removed. Their employment history with the company is also displayed.



Basic Info - edit basic info by changing the information displayed in the text input. save any changes by clicking the save button.

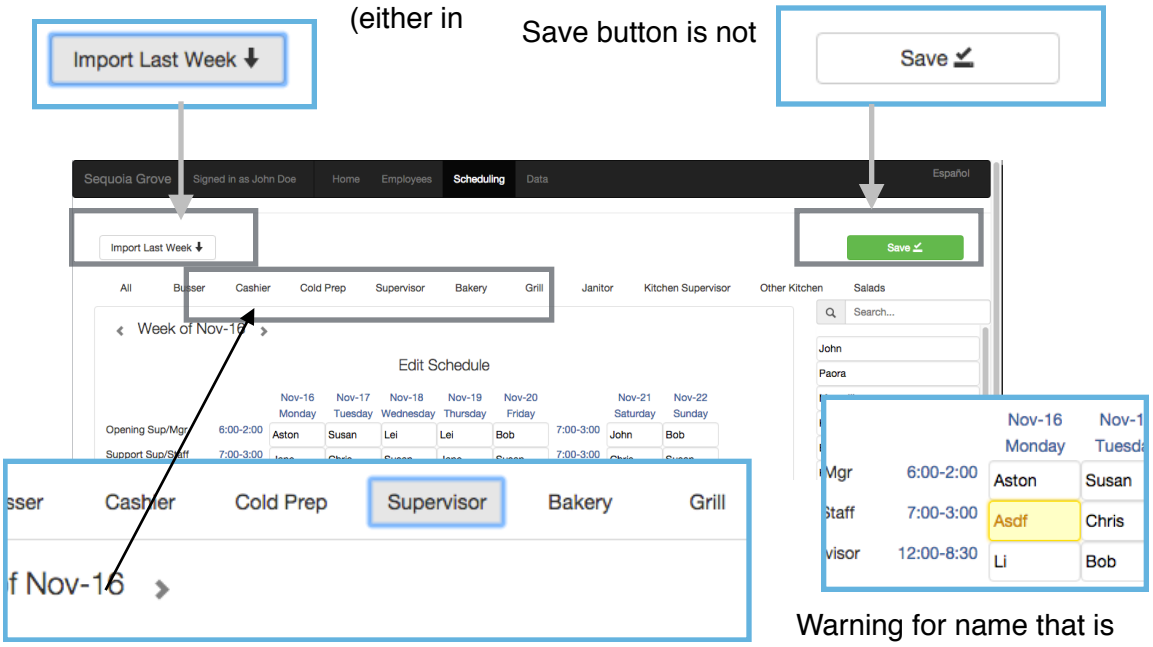


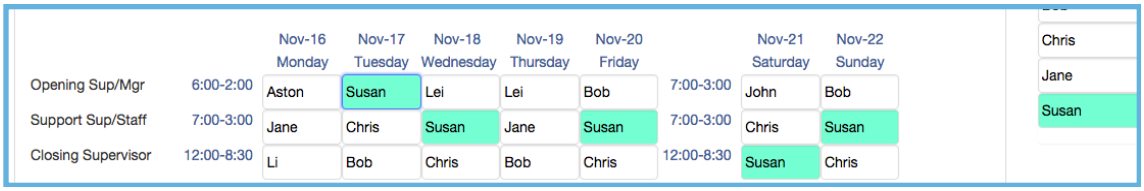
Availability - Add availability by selecting a new availability and adding it with the green + button, remove availability by clicking the red X button next to the corresponding availability.



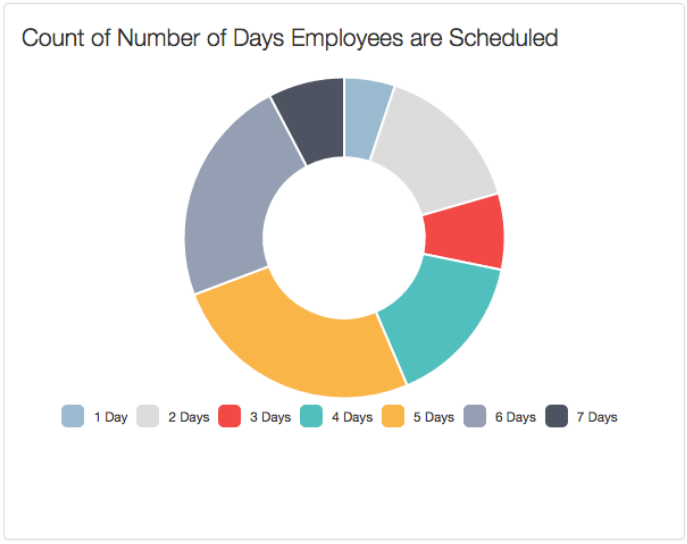
The **Schedule Edit UI** allows the user to create or update a schedule. The week being edited can be changed the same way the schedule view week is changed. The shifts can be narrowed down by type of position by clicking the corresponding position laid out above. Changes made can be saved by clicking the save button. The save button is green when there are no new changes, and default color when there are changes. A new schedule can be populated by importing the previous week relative to the new week. When a schedule is imported, it is automatically saved.

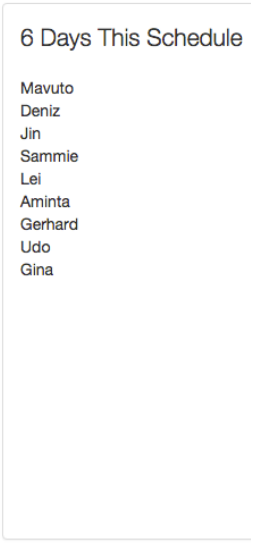
The UI shows feedback when editing the schedule. When a name is clicked (either in

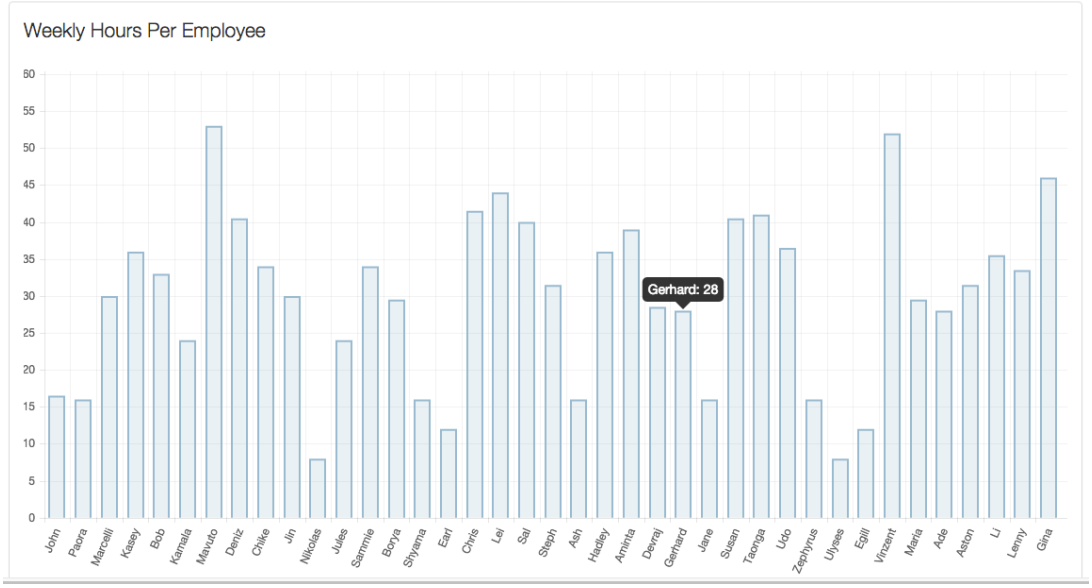




The **Data UI** shows calculations for the current week. The count of days that each employee is scheduled for is calculated. The graph can be clicked to show the names that fall under this category. The count of the number of hours







Section 5.4 Code Description

The front end and back end of this web application involve a large amount of code that required careful planning and studying to create. This section describes how the code was developed, overviews the major components of the code itself, describes the code's output in the GUI, and discusses our methods used for learning new tools and languages.

5.4.1 Major Steps of Designing a User Interface

Designing a user interface begins with clarifying the purpose of the application.
This application was built as a scheduling program, so it needed to be able to display and edit a schedule as well as company employees and their availability.

Next, the views are determined by the logical organization of the purpose.
The views for this project were determined to be as follows:
one to display the schedule
one to edit the schedule
one to edit employees.

Once the views are decided, they are sketched out on paper or a modeling program.
Our layouts for each of these views were drawn in an online modeling program.

Next, the views are coded in HTML, Javascript, and CSS using dummy data hardcoded in the javascript as placeholders in JSON (JavaScript Object Notation) format.

After the front end is set up with dummy data, the back end is constructed to represent the JSON objects. This end holds methods that interact with the database to create objects populated with actual data, which will be sent to the front end.

The front end is then hooked to the back end to call methods to send and receive data.

Inconsistencies with using real data are polished on the front end, and the dummy data is removed. Any necessary debugging is done, and front end features are finally fleshed out completely.

5.4.2 Description of Major Classes

The major classes can be divided by front and back end functionality. The back end has two categories of classes: controllers and models. The models are Java classes used to represent objects that can be put in lists to be sent to the front end. The controllers are Java classes that implement JDBC (Java Database Connection) to interact with the database.

Back End

Java Model Classes

Availability - start and end times for availability separated into hours and minutes.
Weekly Availability - list of Availability for each day of the week
Employment History - start and end dates
Shift - shift id, position id, task name and position name for a shift
Position - the position id, title and location for a position
Employee - the employee's basic information, as well as a list of their positions and employment history, and their weekly availability
Day - the weekday title, name and id of an employee that is specific to a date
Schedule Template - the information relevant to a row in the schedule including the shift information and each day in the week

Back End

Java Controller Classes - data access for each category

Main Controller
Employee Controller
Position Controller
Schedule Controller
Shift Controller

Front End

View Pages - correspond to the views the user can see

Home - schedule view

Employee - edit employee view

Schedule - edit schedule view

Request - data request display

Front End

Javascript - data access and operations specific to each corresponding view

Main

Employee

Home

Schedule

Request

5.4.3 Major features of GUI Programs

The main focus of this web application was ease of usability. This meant making sure there was a low learning curve to using the application by designing the views, and what they did, in a way that was intuitive. The purpose views and their interfaces can be discerned by simply looking at them.

Ensuring ease of usability also meant making sure tasks the user performs are time efficient by automating anything that would help the user perform that task. In the weekly schedule view, for both editing and viewing, switching between weeks is as simple as clicking one button; dates do not need to be typed. The user can narrow down which shifts are by their own id, positions of interest, or locations of interest, with only 1-2 clicks. This allows managers to easily schedule shifts of interest, and employees to easily view their own shifts. In both the employee view and schedule edit view users can search by employee, which allows managers to quickly find an employee's information or which shifts an employee is scheduled. All instances of an employee's name are also highlighted when his/her name is clicked in the schedule edit, allowing managers to easily locate when an employee has been scheduled during the week. Warnings and errors are also shown when attempting to schedule employees that do not exist, or during shifts when they are not available. Finally graphs that summarize the number of days and hours employees work for each week are available to provide a quick summary of employees to managers.

5.4.4 Learning Development Tools

The knowledge to use the tools for this application was gained through prior experience and research on the web. We had prior experience with the angular js web framework and bootstrap, but spring was new to us. Aside from the frameworks, the code for each of the components was written in the plain text editor vim. The integration of the angular and bootstrap front end, spring back end, and jdbc connection to the database was a new setup using familiar tools. We set up the project as a skeleton with the front and back end that was in the directory structure required by the build tool we were using: Maven. Maven is a command line tool that compiles the back end and has a plugin that runs a local server for the front end. Once the blank project ran, we added as much as we could to the front end and set it up. Next, we learned how to make an HTTP request so the front end could request data from the back end. We tested the HTTP request with dummy data. Finally, we integrated JDBC with our current setup and used the example hotel database to practice using JDBC to send SQL queries to the database. Next, we learned how to write more detailed SQL to construct our own tables, and write our own stored procedures and functions for the database. Then, we learned how to use JDBC to retrieve the resultset from the database, construct Java objects, and send the data back to the front end. Each step of the process was a challenge that was overcome. It was a rewarding experience seeing each step of the process successfully run.

Section 5.5 Designing and Implementing a Database

This section summarizes the major steps involved in constructing a database, as well as the work done on this project for each of these steps.

There are five major steps involved in designing and implementing a database:

Information Gathering

Conceptual Database Model

Logical Database Model

Physical Database Design

Front End Design

5.5.1 Information Gathering

To begin designing any database, the design requirements for the database must be determined. The first and most important step is to carefully observe the end users the database is being designed for in order to fully understand the problems these users currently face, and which of these problems can be solved using a database, if any. Before even beginning to model the database, the purpose and requirements of the database must be clearly thought out and understood, or there is no reason to build it. To this end, several information gathering techniques are used to understand the needs of the end users. The following is a list of information gathering techniques commonly used when designing a database for a business:

Business Documentation
On-Site Interviews
Job Shadowing
Surveys
Work Experience

Each of these techniques is a trade-off between the time taken and the understanding of the business provided. Surveys are often the quickest to implement, but provide limit the scope of issues within the business that may be seen to the questions being asked. Business documentation is more time-consuming to read through, but provides a more thorough understanding of the business

structure, and thus where possible issues may originate. On-site interviews are usually even more time consuming, but have much more potential to reveal the issues that employees need solved. Job shadowing helps understand specific issues employees face in their job, and which in turn can help categorize employees in terms of their needs, helping form user groups later on. Work experience is the most effective information gathering technique in terms of the understanding of the business provided, however it is not feasible unless the person(s) being asked to construct the database have already been employees.

The primary information gathering technique we used was years of prior work experience. This gave us a familiarity the information issues faced by the company, from which we derived managers were going through too much trouble to develop a weekly work schedule for their employees. Managers were currently writing schedules out using pencil, paper, and excel spreadsheets, and had to manage each employee's availabilities and positions worked using the same method. Much of this information could be maintained, updated, and viewed much more efficiently using a database and well-planned front end, which was the motivation for creating this scheduling database.

5.5.2 Conceptual Model

Once enough information has been gathered, the next step is to construct a conceptual model of the database based on the issues it is meant to solve. The model is meant to translate the issues that need to be resolved into information objects that contain the data necessary to resolve those issues. The most widely used conceptual model is the Entity-Relationship Model, or ER Model, which models facts in database miniworld as entities that are connected by relationships. Entities are described by the attributes and sub-entities they contain, while relationships are described the entities they relate, their cardinality between entities, and potentially any descriptive fields they may contain. This helps designers visualize the issues being addressed as objects that correspond to one another through relationships, allowing a more clear understanding of how information will flow through the database.

In order to determine how an employees should be scheduled, several entities needed to be defined. Firstly the 'Employee' entity was needed to track each employee, a 'Position' entity was needed to track each possible position, and the 'Shift' entity was needed to track each time shift an employee may work. The 'Shift' entity was later split into two subclass, 'Weekday Hours' and 'Weekend Hours', since the shifts available on weekdays are different from those during the weekends. These primary classes were then connected using three relationships, employees with position by 'Has_Position', position with shifts by 'Has_Shifts', and employees with shifts by 'Is_Scheduled_For'. This formed

the basis of tracking which employees may work which shifts based on their position, and which shifts they are actually scheduled for. Additional entities were then added to track when employees are available, times employees have requested off, which employees cannot with together, and login information. After connecting these entities with the appropriate relationships, we had our basic ER Model.

5.5.3 Logical Model

Next the Conceptual Model is converted to the Logical Model, which models that database as it will be represented in the computer. Almost all Logical Models are represented using the Relational Model, which describes all facts in the database as relations that connect to one another via referential keys. Relations are sets of tuples whose values are an ordered set of attributes unique to each relation. They can thought of as tables whose columns are the attributes describing the relation, and whose rows are individual entries, or facts, that have been added to the database. Each relation must have a primary key defined, which is minimal set of attributes that can uniquely identify a tuple within that relation. This allows for a method of finding individual facts within every relation. To convert the Conceptual Model to the Logical Model, hierarchies of entities are flattened into individual relations, composite attributes containing sub-attributes are separated into individual attributes or moved into their own relation, and relationships are removed by implementing referential key attributes that refer to the primary key attributes of related entities.

To convert our ER Model to a Relational Model, the Employee and Position class were converted directly into relations. The Shift entity was converted to a strong entity with its own id as a primary key, and Weekday_Hours and Weekend_Hours were converted to weak entities with without primary keys. These three entities were then converted into relations by including Shift's 'id' in Weekday_Hours and Weekend_Hours as foreign keys. Next the relationships connecting these relations were converted. Has_Postition and Is_Scheduled_For became relations using the Merged Key Method, and Has_Shifts was eliminated using the Foreign Key Method. The remaining relations were converted directly from their corresponding entities, and their relationships were converted using the Foreign Key Method. This left us with our Relational Model, which we could use to start defining queries and convert to a physical database.

After a Logical Model has been designed, the database can be cleanly implemented into a physical database on a server. Although modifications will invariably be made as more effective ways of storing data a realized, modeling the database in the Logical Model greatly reduces these instances

by providing a thorough picture of the physical database beforehand. In order to further reduce modifications that may be needed later to eliminate redundancy in data, it is best to normalize all relations in the Relational Model into at least 3rd Normal Form. We made sure to normalize our relations before implementing them on a server.

5.5.4 Information Gathering

The physical model is implemented by selecting a database management system, or DBMS, that the server will use to manage the relations being implemented as tables, as well as respond to requests for inserting, deleting, modifying, or viewing data. This step involves learning whatever language the DBMS provides for implementing procedural code in the database, in addition to SQL for creating the database itself. It is also useful to learn about any special objects or classes provided by the DBMS that may help speed up queries, minimize storage, and automate certain tasks. Most DBMS's provide a method for storing queries to the database in a function, so that the query may be automated with different parameters. They usually also allow queries to be stored as views, which are virtual tables that contain combinations of data that is often required together by front end users. These views help speed up writing calls to the database from the back end application that connects that database to the front end.

For the Databases class of Fall 2015, CMPS 342, our database was implemented on Delphi, the student database server for the Computer Science Department at California State University, Bakersfield. This server uses Oracle as its DBMS, so our database was implemented using PL/SQL. Our first step was to make SQL CREATE and DROP statements for all of relations, and then make scripts to automate the process of adding, dropping, and reseting these relations. Next a series of INSERT statements were made for each table to check constraints. Once dummy data was set up and checked against tables, more advanced objects were created to help automate the process of querying from the database. Views were created for common queries, functions were created to pass parameters to other queries, and packages were created to allow for complex parameters.

5.5.5 Front End Design

Once the physical database has been filled with sample data, the final step is to connect the database to the front end of the application where the users can access the data. This involves creating the back end of the application that is responsible for making queries to the database,

organizing the data into the format desired by the front end, and passing the data along to the front end. The back end is also responsible for listening for requests from the front end to view or modify data in the database, reformatting the request into the format desired by the DBMS, the passing the request to the database.

The front end is responsible for all interaction a user has with the database. Any requests to view, add, delete, or modify anything from the database are acquired by the front end. This involves designing a GUI that is simple and intuitive to understand by the user, and does not confuse the user with unnecessary details. The GUI presents the user with the information they are expected to need for solving the issues the database is designed to solve, and presents the users with a method for altering data or inserting data to update the database in regards to those issues.

Determining the combination of information that is most useful to the user is often the most difficult part of designing a front end, and must therefore be planned out in advance. In our case the different screens, or views, users were to be presented with were planned out in advance by modeling them as wireframes. These wireframes were then coded into actual GUI pages with fields for displaying data and reports pulled from the database. Next fake information was created to determine what format the front end needed for its data. Finally data was given to the front end in that format from the back end, which had been connected to the database.

Our web application utilized several tools and components. The back end was written in Java and used JDBC through the Spring framework. The front end was written in HTML, CSS, and Javascript. It makes use of AngularJS javascript web application framework (https://angularjs.org/), and Bootstrap CSS framework for styling (http://getbootstrap.com/css/). The back end uses the Spring Java framework (http://projects.spring.io/spring-framework/). Web libraries were installed as dependencies for added functionality. Moment js (http://momentjs.com/) is a date time library that was used for time formatting and calculations on the front end. Angular Charts (http://jtblin.github.io/angular-chart.js/) was used to display graphs and charts.

Documentation and current code can be viewed on github (online code version control and sharing) at: **https://github.com/bethgrace5/sequoia-grove**

Embedded Questions

(3b) An ability to analyze a problem, and identify and define the computing requirements and specifications appropriate to its solution.	Bethany Armitage: 9 Jasjot Sumal: 8
(3e) An ability to design, implement and evaluate a computer-based system, process, component, or program to meet desired needs. An ability to understand the analysis, design, and implementation of a computerized solution to a real-life problem.	Bethany Armitage: 9 Jasjot Sumal: 8
(3f) An ability to communicate effectively with a range of audiences. An ability to write a technical document such as a software specification white paper or a user manual.	Bethany Armitage: 8 Jasjot Sumal: 9
(3j) An ability to apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer-based systems in a way that demonstrates comprehension of the tradeoffs involved in design choices.	Bethany Armitage: 9 Jasjot Sumal: 9

Bibliography

The section describes references used throughout the paper. The citations are formatted using MLA standards.

“4 Developing and Using Stored Procedures.” Database 2 Day Developer’s Guide. Oracle. Web. 11 Nov. 2015.
<https://docs.oracle.com/cd/B28359_01/appdev.111/b28843/tdddg_procedures.htm#CIHGAGJG>

“Advantages of Subprograms.” PL/SQL User’s Guide and Reference. Release 2(9.2)
Oracle. Web.
12 Nov. 2015
<https://docs.oracle.com/cd/A97630_01/appdev.920/a96624/08_subs.htm>

"Batches of SQL Statements." Batches of SQL Statements. Microsoft. Web. 3 Nov. 2015.
<https://msdn.microsoft.com/en-us/library/ms712553%28v=vs.85%29.aspx>

"Ch 5. Schema Objects." Database Concepts. Oracle. Web. 4 Nov. 2015.
<http://docs.oracle.com/cd/B19306_01/server.102/b14220/schema.htm#i5663>

Chen, Peter. "The Entity-Relationship Model - Toward a Unified View of Data." *ACM Transactions on Database Systems* 1.1 (1976): 9-36. *Fakultät Für Mathematik Und Informatik*. Association for Computing Machinery, Inc. Web. 14 Oct. 2015. <http://www.minet.uni-jena.de/dbis/lehre/ws2005/dbs1/Chen.pdf>.

Connolly, Thomas, and Carolyn Begg. "Chapter 10: Fact-Finding Techniques." *Database Systems*. 4th ed. Essex, England: Pearson Education Limited, 2005. 314-341. Print.

“Controlling Program Flow.” Database 2 Day Developer’s Guide. Oracle. Web.
12 Nov. 2015
<https://docs.oracle.com/cd/B28359_01/appdev.111/b28843/tdddg_procedures.htm#TDDDG44000>

"MySQL." Wikipedia. Wikimedia Foundation. Web. 13 Nov. 2015.
<https://en.wikipedia.org/wiki/MySQL>

“PL/SQL Triggers.” PL/SQL Tutorial. Web. 11 Nov. 2015.
<http://plsql-tutorial.com/plsql-triggers.htm>

“Retrieving Data from a Set Using Cursors and Cursor Variables.” Database 2 Day Developer’s Guide. Oracle. Web.
12 Nov. 2015
<https://docs.oracle.com/cd/B28359_01/appdev.111/b28843/tdddg_procedures.htm#TDDDG45000>

"SQL." Wikipedia. Wikimedia Foundation. Web. 3 Nov. 2015.
<https://en.wikipedia.org/wiki/SQL>

"SQL*Plus Quick Start." SQL*Plus Quick Start. Oracle. Web. 3 Nov. 2015.
<http://docs.oracle.com/cd/B19306_01/server.102/b14357/qstart.htm>

"Transact-SQL." Wikipedia. Wikimedia Foundation. Web. 13 Nov. 2015.
<https://en.wikipedia.org/wiki/Transact-SQL>

“Triggers.” Database Concepts. Oracle. Web. 11 Nov. 2015.
<https://docs.oracle.com/cd/B28359_01/server.111/b28318/triggers.htm>

“Using Composite Data Structures; Records.” Database 2 Day Developer’s Guide. Oracle. Web.
12 Nov. 2015
<https://docs.oracle.com/cd/B28359_01/appdev.111/b28843/tdddg_procedures.htm#TDDDG43000>

“Using Variables and Constants.” Database 2 Day Developer’s Guide. Oracle. Web.
12 Nov. 2015
<https://docs.oracle.com/cd/B28359_01/appdev.111/b28843/tdddg_procedures.htm#TDDDG42000>

"When Should I Use Oracle's Index Organized Table? Or, When Shouldn't I?" APC. Stack Overflow.
Stack Exchange. Web. 4 Nov. 2015.
<http://stackoverflow.com/questions/3382939/when-should-i-use-oracles-index-organized-table-or-when-shouldnt-i>