Project for Database Design

Phase IV. Documentation

Qianying Lin Shasha Jin

(Week 11-16: Mar.31-May.5)

0. Pre-Illumination

In this project report we will follow the requirement of Phase IV directly. In Section 1 we gave problem description copied from Web site; in Section 2 we answered 3 questions listed in the project and justified our solution; in Section 3 we exhibited EER diagram with all assumptions; in Section 4 we showed our relational schema after normalization; in Section 5 we gave all requested SQL statements for both views and queries; and in Section 6 we gave dependency diagram induced from relational schemas. Finally, a short summary is given at the end of this report.

1. Problem Description

2. Three Questions

2.1 1. Is the ability to model super-class / subclass relationships likely to be important in such environment? Why or why not?

Yes, super-class/subclass relationships are important.

First, these relationships can reduce redundancy by dealing with these situations: 1) certain attributes may apply to some but not all entities of the superclass; 2) some relationship types may be participated in only by entities that are members of the subclass. For example, if only a few “Student” type entities have access to “Lab” entity type entities, we can use subclass to deal with these specific subclass ”student” entities, rather than the whole “Student” type entities. Second, these relationships implement extensibility better. For example, “Student” is the super-class of “CS student” and “GIS student”. If we want to add “access to Lab” to both “CS student” and “GIS student”, we can only add this property to “Student” rather than edit them separately.

2.2 2. Can you think of 5 more rules (other than the one explicitly described above) that are likely to be used in a school environment? Add your rules to the above requirement to be implemented.

Rule 1: Each employee can only purchase at most one lot.
Rule 2: Loan companies can be either commercial banks or credit unions. We record the bank ID and the interest rate for each loan company.
Rule 3: First_name and Last_name for every employee should be different.
Rule 4: The number of floor plan for each subdivision is no more than 20.
Rule 5: The term for loan_applicant is no less than 10 years.

2.3 Justify using a Relational DBMS like Oracle for this project.

Controlling redundancy: redundancy results from storing the same data multiple times. DBMS used controlled redundancy to ensure the consistency and save space.
Restricting Unauthorized Access: A DBMS should provide a security and authorization subsystem, which the DBA uses to create accounts and to specify

account restrictions because most users will not be authorized to access all information in the database

Providing Persistent Storage for Program Objects: the DBMS software automatically performs any necessary conversions that convert objects with these complex structures into a format suitable for file storage. This can also avoid impedance mismatch problem.

Providing Storage Structures and Search Techniques for Efficient Query Processing: DBMS uses provide specialized data structure, search techniques, and auxiliary files (indexes) to speed up disk search for the desired records. DBMS also has a buffering or caching module that maintains parts of the data-base in main memory buffers.

Providing Backup and Recovery: The backup and recovery subsystem of the DBMS is responsible for recovery.

Providing Multiple User Interfaces: DBMS should provide a variety of user interfaces for users with varying levels of technical knowledge use a data- base.

Representing Complex Relationships among Data: A DBMS must have the capability to represent a variety of complex relationships among the data, to define new relationships as they arise, and to retrieve and update related data easily and efficiently.

Enforcing Integrity Constraints: Most database applications have certain integrity constraints that must hold for the data. A DBMS should provide capabilities for defining and enforcing these constraints

Permitting Inferencing and Actions Using Rules: associate triggers with tables. A trigger is a form of a rule activated by updates to the table, which results in performing some additional operations to some other tables. More powerful functionality is provided by active database systems, which provide active rules that can automatically initiate actions when certain events and conditions occur.

Additional Implications of Using the Database Approach: Potential for Enforcing Standards. Reduced Application Development Time. Flexibility. Availability of Up-to-Date Information. Economies of Scale.

3. EER diagram with all assumptions

1 Employees can only works for one subdivision.

2 Subdivisions can have 0 to N employees.

3 Each subdivision has at most one manager.

4 Subdivision has at least one floor plan.

5 Subdivision has at least one lot.

6 Subdivision has at least one inventory.

7 0 to N customers can visit the same subdivision.

8 Each customer can visit N subdivisions.

9 One customer can have tour to N subdivisions.

10 0 to N customers can have tours to the same subdivision.

11 One employee can at most tour N customers.

12 One contract can be signed by only one customer at most.

13 One subdivision can sign contracts with N customers.

14 Employee can prepare 0 to N contracts.

15 One contract can be prepared by only one employee.

16 Loan applicants can be checked by at most N companies.

17 Loan companies can check 0 to N loan applicants.

4. Relational Schema in Third Normal Form

4.1 Relational Schema

Put your Relational Schema diagram and assumptions here.

4.2 Format for Every Relation

Put your format here.

5. All Requested SQL Statements

5.1 Creation of Database with SQL Statements

5.1.1 Table Creation

Using SQL statement, we created tables as follows:

3.1.1.1 Loan company table

```
CREATE TABLE LOAN_COMPANY
(Name VARCHAR(20) NOT NULL,
PRIMARY KEY (Name))
```

3.1.1.2 Loan applicant table

```
CREATE TABLE LOAN_APPLICANT
(Loan_ID CHAR(5) NOT NULL,
Credit_report CHAR(5),
A_D_data CHAR(10),
Term INT,
Type VARCHAR(20),
PRIMARY KEY (Loan_ID),
CONSTRAINT chk_term CHECK (Term < 100))
```

3.1.1.3 Check_loan table

```
CREATE TABLE CHECK_LOAN
(Loan_ID CHAR(5) NOT NULL,
Company_name VARCHAR(20) NOT NULL,
PRIMARY KEY (Company_name, Loan_ID),
FOREIGN KEY (Company_name) REFERENCES LOAN_COMPANY(Name),
FOREIGN KEY (Loan_ID) REFERENCES LOAN_APPLICANT(Loan_ID))
```

3.1.1.4 Customer

```
CREATE TABLE CUSTOMER
(Email VARCHAR(50) NOT NULL,
First_name VARCHAR(20) NOT NULL,
Last_name VARCHAR(20) NOT NULL,
Income INT,
```

```
Credit_score INT,
Num_of_bath INT,
Num_of_bedroom INT,
School_rank INT,
Square_footage INT,
Customer_type VARCHAR(15),
Loan_ID CHAR(5),
PRIMARY KEY (Email),
CONSTRAINT chk_credit CHECK (Credit_score <= 850 and Credit_score >= 300),
CONSTRAINT chk_bath CHECK (Num_of_bath >= 1 and Num_of_bath <= 5),
CONSTRAINT chk_bed CHECK (Num_of_bedroom >= 1 and Num_of_bedroom <= 5),
CONSTRAINT chk_square CHECK ( Square_footage < 10000),
CONSTRAINT chk_custo_type CHECK (Customer_type IN ('potential', 'existing')),
FOREIGN KEY (Loan_ID) REFERENCES LOAN_APPLICANT(Loan_ID))
```

3.1.1.5 Phone num

```
CREATE TABLE PHONE_NUM
(Email VARCHAR(50) NOT NULL,
Phone_num CHAR(13),
CONSTRAINT pk_phone PRIMARY KEY (Email, Phone_num),
FOREIGN KEY (Email) REFERENCES CUSTOMER(Email))
```

3.1.1.6 Subdivision

```
CREATE TABLE SUBDIVISION
(Name VARCHAR(20) NOT NULL,
Start_year CHAR(4),
ZIP_zode CHAR(5),
School_rank INT,
```

PRIMARY KEY (Name))

3.1.1.7 Floor plan

CREATE TABLE FLOOR_PLAN
(Sub_name VARCHAR(20) NOT NULL,
Floor_num INT NOT NULL,
Name VARCHAR(20) NOT NULL,
Square_footage INT,
Price INT,
Num_of_floor INT,
Num_of_bedroom INT,
Num_of_bath INT,
Num_of_garage INT,
CONSTRAINT pk_floor PRIMARY KEY (Sub_name, Floor_num),
FOREIGN KEY (Sub_name) REFERENCES SUBDIVISION (Name),
CONSTRAINT chk_bath1 CHECK (Num_of_bath >= 1 and Num_of_bath <= 5),
CONSTRAINT chk_bed1 CHECK (Num_of_bedroom >= 1 and Num_of_bedroom <= 5),
CONSTRAINT chk_floor1 CHECK (Num_of_floor >= 1 and Num_of_floor <= 3),
CONSTRAINT chk_garage1 CHECK (Num_of_garage >= 1 and Num_of_garage <= 3),
CONSTRAINT chk_floornum1 CHECK (Floor_num < 10000),
CONSTRAINT chk_square1 CHECK (Square_footage < 10000))

3.1.1.8 Lot

CREATE TABLE LOT
(Sub_name VARCHAR(20) NOT NULL,
Lot_num CHAR(5) NOT NULL,
Lot_Size INT,
Facing CHAR(1),
Floor_num INT,

CONSTRAINT pk_lot PRIMARY KEY (Sub_name, Lot_num),
CONSTRAINT chk_floornum2 CHECK (Floor_num < 10000),
CHECK(Facing IN ('N', 'S', 'W', 'E')),
FOREIGN KEY (Floor_num, Sub_name) REFERENCES FLOOR_PLAN (Floor_num, Sub_name))

3.1.1.9 Employee

create table EMPLOYEE
(Employee_ID CHAR(11) NOT NULL,
First_name VARCHAR(20),
Last_name VARCHAR(20),
Phone_num CHAR(13),
Salary INT,
Year_of_experience INT,
Loan_ID CHAR(5),
Sub_name VARCHAR(20),
PRIMARY KEY (Employee_ID),
CONSTRAINT chk_experience CHECK (Year_of_experience <= 55 AND Year_of_experience >= 0),
FOREIGN KEY (Loan_ID) REFERENCES LOAN_APPLICANT(Loan_ID),
FOREIGN KEY (Sub_name) REFERENCES SUBDIVISION (Name))

3.1.1.10 Inventory

CREATE TABLE INVENTORY
(Sub_name VARCHAR(20),
Inventory_id CHAR(5),
Price INT,
Move_in_date CHAR(10),
Lot_num CHAR(5),
Availability char(1),
Floor_num INT,

```
CONSTRAINT pk_inventory PRIMARY KEY (Sub_name, Inventory_id),
FOREIGN KEY (Floor_num, sub_name) REFERENCES FLOOR_PLAN (Floor_num, sub_name),
FOREIGN KEY (sub_name, Lot_num) REFERENCES LOT(Sub_name, Lot_num),
CONSTRAINT  ava_check CHECK (Availability in ('T','F')),
CONSTRAINT chk_floornum3 CHECK (Floor_num < 10000))
```

3.1.1.11 Sales

```
CREATE TABLE SALES_AGENT
(Employee_ID CHAR(11) NOT NULL,
PRIMARY KEY (Employee_ID),
FOREIGN KEY (Employee_ID) REFERENCES EMPLOYEE (Employee_ID))
```

3.1.1.12 Shift

```
CREATE TABLE SHIFT
(Employee_ID CHAR(11) NOT NULL,
Shift char(3),
PRIMARY KEY (Employee_ID, Shift),
FOREIGN KEY (Employee_ID) REFERENCES SALES_AGENT (Employee_ID),
CONSTRAINT shift_check CHECK (Shift in ('Mon','Tue','Wed','Thu','Fri','Sat','Sun')))
```

3.1.1.13 Manager

```
CREATE TABLE MANAGER
(Employee_ID CHAR(11) NOT NULL,
Start_date CHAR(10),
Sub_name VARCHAR(20),
PRIMARY KEY (Employee_ID),
FOREIGN KEY (Sub_name) REFERENCES SUBDIVISION (Name),
FOREIGN KEY (Employee_ID) REFERENCES EMPLOYEE (Employee_ID))
```

3.1.1.14 Secretary

CREATE TABLE SECRETARY
(Employee_ID CHAR(11) NOT NULL,
Hourly_range INT,
PRIMARY KEY (Employee_ID),
FOREIGN KEY (Employee_ID) REFERENCES EMPLOYEE (Employee_ID))

3.1.1.15 Contract

CREATE TABLE CONTRACT
(Contract_ID CHAR(5) NOT NULL,
Sale_price INT,
Date_signed CHAR(10),
Employee_ID CHAR(11),
PRIMARY KEY (Contract_ID),
FOREIGN KEY (Employee_ID) REFERENCES EMPLOYEE (Employee_ID))

3.1.1.16 Associate

CREATE TABLE ASSOCIATE
(Contract_id CHAR(5) NOT NULL,
sub_name VARCHAR(20),
Floor_num INT,
Lot_num CHAR(5),
Primary key (Contract_id, sub_name, Floor_num, Lot_num),
FOREIGN KEY (Contract_id) REFERENCES CONTRACT(Contract_ID),
FOREIGN KEY (Floor_num, sub_name) REFERENCES FLOOR_PLAN (Floor_num, Sub_name),
FOREIGN KEY (sub_name, Lot_num) REFERENCES LOT(Sub_name, Lot_num),
CONSTRAINT chk_floornum4 CHECK (Floor_num < 10000))

3.1.1.17 Visit

CREATE TABLE VISIT
(Customer_email VARCHAR(50) NOT NULL,
Sub_name VARCHAR(20) NOT NULL,
Visit_date CHAR(10),
CONSTRAINT pk_visit PRIMARY KEY (Customer_email, Sub_name),
FOREIGN KEY (Sub_name) REFERENCES SUBDIVISION (Name),
FOREIGN KEY (Customer_email) REFERENCES CUSTOMER (Email))

3.1.1.18 Sign

CREATE TABLE SIGN
(Contract_ID CHAR(5) NOT NULL,
Customer_email VARCHAR(50) NOT NULL,
Sub_name VARCHAR(20) NOT NULL,
PRIMARY KEY (Customer_email, Sub_name, Contract_ID),
FOREIGN KEY (Sub_name) REFERENCES SUBDIVISION (Name),
FOREIGN KEY (Contract_ID) REFERENCES CONTRACT (Contract_ID),
FOREIGN KEY (Customer_email) REFERENCES CUSTOMER (Email))

3.1.1.19 Tour

CREATE TABLE TOUR
(Employee_ID CHAR(11) NOT NULL,
Customer_email VARCHAR(50) NOT NULL,
Sub_name VARCHAR(20) NOT NULL,
Tour_date CHAR(10),
PRIMARY KEY (Customer_email, Sub_name, Employee_ID),
FOREIGN KEY (Sub_name) REFERENCES SUBDIVISION (Name),
FOREIGN KEY (Employee_ID) REFERENCES EMPLOYEE (Employee_ID),
FOREIGN KEY (Customer_email) REFERENCES CUSTOMER (Email))

5.1.2 A Database State

We insert some values into the database in order to test our SQL create view and query statement. Here we just give one example of insertions as follow:

3.1.1.1 Loan company table

INSERTION OF TABLE LOAN_COMPANY (Example)

INSERT INTO LOAN_COMPANY (NAME) VALUES ('Discover');

Table 3.1 shows the states of Loan company database schema.

Name
Chase
Discover
AMEX
Citi
Capital One
BOA
AAA
UA
BA
Delta

3.1.1.2 Loan applicant table

INSERTION OF TABLE LOAN_APPLICANT (Example)

INSERT INTO LOAN_APPLICANT VALUES ('00001', ‘R0001’, ‘10/10/2010’, 10, ‘fixed rate’);

Table 3.2 shows the states of Loan applicant database schema.

Loan_id	Credit_report	A/D date	Term	Type
00001	R0001	10/10/2010	10	fixed rate
00002	R0002	11/12/2000	15	fixed rate
00003	R0003	12/22/2011	20	arm
00004	R0004	10/23/2012	25	fixed rate
00005	R0005	01/13/2011	30	fixed rate
00006	R0006	12/01/2014	35	fixed rate
00007	R0007	11/30/1990	10	fixed rate
00008	R0008	03/12/2011	15	fixed rate
00009	R0009	02/23/2012	20	fixed rate
00010	R0010	04/11/2010	25	arm
00011	R0011	01/21/1987	10	arm
00012	R0012	10/23/2012	15	arm
00013	R0013	08/01/2010	20	arm
00014	R0014	09/05/2011	25	arm
00015	R0015	07/23/2012	10	arm
00016	R0016	01/01/1989	15	arm

3.1.1.3 Check_loan table

INSERTION OF TABLE CHECK_LOAN (Example)

INSERT INTO CHECK_LOAN VALUES ('00001', ‘Chase);

Table 3.3 shows the states of Check Loan database schema.

Loan_id	Company_name
00001	Chase
00002	Discover
00003	AMEX
00004	Citi
00004	Chase
00005	Capital One
00006	BOA
00007	AAA
00007	Chase
00008	UA
00009	Chase
00010	Discover
00011	AMEX
00012	Citi
00013	Chase
00014	Capital One
00015	BOA
00016	AAA

3.1.1.4 Customer

INSERTION OF TABLE CUSTOMER (Example)

INSERT INTO CUSTOMER VALUES ('jbutt@gmail.com', ‘James’, ‘Butt’,

50000, 500, 1, 1, 10, 3000, ‘potential’, ‘00001’);

Table 3.4 shows the states of Customer database schema.

Email	First_name	Last_name	Income	Credit_score	Num_of_bath	Num_of_bedroom	School_rank	Square_footage	Customer_type	Loan_id
jbutt@gmail.com	James	Butt	50000	500	1	1	10	3000	potential	00001
josephine_darakjy@darakjy.org	Josephine	Darakjy	40000	550	2	2	20	2000	potential	00002
art@venere.org	Art	Venere	30000	750	3	3	30	1000	existing	00003
lpaprocki@hotmail.com	Lenna	Paprocki	30000	600	4	4	40	5000	potential	00004
donette.foller@cox.net	Donette	Foller	60000	730	5	5	50	3000	existing	00005
simona@morasca.com	Simona	Morasca	50000	790	2	2	20	2000	potential	00006
mitsue_tollner@yahoo.com	Mitsue	Tollner	40000	700	5	5	30	2000	existing	00007
leota@hotmail.com	Leota	Dilliard	30000	800	3	3	50	2000	potential	00008
sage_wieser@cox.net	Sage	Wieser	50000	730	1	1	20	2000	potential	00009
kris@gmail.com	Kris	Marrier	40000	750	3	1	10	2000	existing	Null
minna_amigon@yahoo.com	Minna	Amigon	50000	740	3	2	30	2000	potential	Null
amaclead@gmail.com	Abel	Maclead	40000	760	2	3	50	3000	potential	Null
kiley.caldarera@aol.com	Kiley	Caldarera	30000	750	3	4	20	3000	existing	Null
gruta@cox.net	Graciela	Ruta	30000	710	3	5	20	3000	existing	Null
calbares@gmail.com	Cammy	Albares	60000	770	2	2	30	3000	potential	Null
mattie@aol.com	Mattie	Poquette	50000	730	5	5	20	3000	existing	Null
meaghan@hotmail.com	Meaghan	Garufi	40000	700	1	3	30	3000	existing	Null
gladys.rim@rim.org	Gladys	Rim	30000	730	5	1	50	3000	potential	Null
yuki_whobrey@aol.com	Yuki	Whobrey	50000	750	1	5	30	4000	existing	Null

3.1.1.5 Phone num

INSERTION OF TABLE Phone num (Example)

INSERT INTO Phone num VALUES ('jbutt@gmail.com', ‘504-621-8927’);

Table 3.5 shows the states of Phone num database schema.

Email	Phone_num
jbutt@gmail.com	504-621-8927
josephine_darakjy@darakjy.org	810-292-9388
art@venere.org	856-636-8749
art@venere.org	907-385-4412
lpaprocki@hotmail.com	513-570-1893
donette.foller@cox.net	419-503-2484
simona@morasca.com	773-573-6914
mitsue_tollner@yahoo.com	408-752-3500
leota@hotmail.com	605-414-2147
sage_wieser@cox.net	410-655-8723
sage_wieser@cox.net	215-874-1229
kris@gmail.com	631-335-3414
minna_amigon@yahoo.com	310-498-5651
amaclead@gmail.com	440-780-8425
kiley.caldarera@aol.com	956-537-6195
gruta@cox.net	602-277-4385
calbares@gmail.com	931-313-9635
mattie@aol.com	414-661-9598
mattie@aol.com	313-288-7937

meaghan@hotmail.com	815-828-2147
gladys.rim@rim.org	610-545-3615
yuki_whobrey@aol.com	408-540-1785

3.1.1.6 Subdivision

INSERTION OF TABLE Subdivision (Example)

INSERT INTO Subdivision VALUES (‘Spring Valley’, ‘2000’, ‘99501’, 10);

Table 3.6 shows the states of Subdivision database schema.

Name	Start_year	ZIP_code	School_rank
Spring Valley	2000	99501	10
Richardson	2001	45011	2
Dallas	2002	44805	1
Irvine	2003	60632	43
Jeffson creek	2004	95111	55
Gainesville	2005	57105	3
Orlando	2006	21224	5
Jacksonville	2007	19443	8
High land park	2008	11953	4
South lake	2009	90034	9

3.1.1.7 Floor plan

INSERTION OF TABLE Floor plan (Example)

INSERT INTO Floor plan VALUES (‘Spring Valley’, 0001, ‘Green Home’, 2000, 110000, 1, 1, 1, 3);

Table 3.7 shows the states of Floor plan database schema.

Sub_name	Floor_num	Name	Square_footage	Price	Num_of_floor	Num_of_bedroom	Num_of_bath	Num_of_garage
Spring Valley	0001	Green Home	2000	110000	1	1	1	3
Richardson	0002	Affordable House	3000	220000	2	2	4	3
Dallas	0001	Cape Cod House	4000	330000	3	3	3	2
Dallas	0002	Classic Revival	5000	440000	1	4	2	1
Irvine	0005	Colonial House	3000	550000	2	2	1	1
Jeffson creek	0006	Cottage House	3500	660000	3	5	2	2
Gainesville	0007	Country House	5000	770000	3	2	3	3
Orlando	0008	Craftsman Style	2500	880000	2	3	4	2
Orlando	0009	Custom House	2500	990000	1	4	2	3
Jacksonville	0010	Colonial House	3000	330000	1	5	2	1
Jacksonville	0011	Cottage House	3500	440000	2	1	5	2
High land park	0012	Country House	5000	550000	3	2	2	3
South lake	0001	Craftsman Style	2500	660000	1	3	3	3
South lake	0002	Affordable House	3000	100000	2	4	4	2
South lake	0003	Cape Cod House	4000	1000000	1	5	5	1

3.1.1.8 Lot

INSERTION OF TABLE Lot (Example)

INSERT INTO Lot VALUES (‘Spring Valley’, ‘0001’, 3000, ‘S’, 00001);

Table 3.8 shows the states of Lot database schema.

Sub_name	Lot_num	Size	Facing	Floor_num
Spring Valley	00001	3000	S	0001
Spring Valley	00002	3000	N	0001
Richardson	00001	4000	E	0002
Dallas	00004	5000	W	0001
Dallas	00005	6000	S	0002
Irvine	00006	4000	N	0005
Jeffson creek	00007	4500	S	0006
Gainesville	00008	6000	N	0007
Orlando	00009	3500	E	0008
Orlando	00010	3500	S	0009
Orlando	00011	4000	S	0009
Jacksonville	00012	4000	S	0010
Jacksonville	00013	4500	W	0011
High land park	00014	6000	W	0012
South lake	00015	3500	N	0001
South lake	00016	4000	E	0002
South lake	00017	5000	W	0003

3.1.1.9 Employee

INSERTION OF TABLE Employee (Example)

INSERT INTO Employee VALUES (‘000-00-0000‘, ‘Emily’, ‘Navathe’, ‘214-456-7626’, 50000, 3, ‘00010’,‘Spring Valley’);

Table 3.9 shows the states of Employee database schema.

Employee_id	First_name	Last_name	Phone_num	Salary	Year_of_experience	Loan_id	Sub_name
000-00-0000	Emily	Navathe	214-456-7626	50000	3	00010	Spring Valley
111-11-1111	Tom	Brown	214-369-8759	40000	4	00011	Richardson
222-22-2222	Jimmy	Johnson	469-765-9754	30000	5	00012	Dallas
333-33-3333	Sally	Smith	214-436-6336	30000	3	00013	Irvine
444-44-4444	Jeniffer	Smack	214-567-4767	60000	6	00014	Jeffson creek
555-55-5555	Smuel	Sunder	972-456-2552	50000	5	00015	Gainesville
666-66-6666	Raja	Farage	972-832-9317	40000	2	00016	Orlando
777-77-7777	Kenneth	Chenault	214-134-8643	30000	6	null	Jacksonville
888-88-8888	Brett	Cotton	469-295-3694	50000	3	null	High land park
999-99-9999	Adam	Daley	469-478-3688	40000	4	null	South lake
101-01-0101	George	Cobb	469-658-3978	50000	6	null	Gainesville
121-21-2121	Ivor	Page	972-843-6823	40000	3	null	Orlando
131-31-3131	Joseph	Tomason	972-987-9843	30000	5	null	Jacksonville
141-41-4141	Sara	Gaddis	972-345-9734	20000	3	null	High land park

3.1.1.10 Inventory

INSERTION OF TABLE Inventory (Example)

INSERT INTO Inventory VALUES (‘‘Spring Valley’, ‘00016’, 120000,

‘10/10/2014’, ‘00001’, ‘T’, 1);

Table 3.10 shows the states of Inventory database schema.

Sub_name	Inventory_id	Price	Move_in_date	Lot_num	Availability	Floor_num
Spring Valley	00016	120000	10/10/2014	00001	T	0001
Spring Valley	00015	230000	11/11/2015	00002	F	0001
Richardson	00014	340000	12/21/2014	00001	T	0002
Dallas	00013	450000	10/23/2016	00004	T	0001
Irvine	00012	560000	01/11/2013	00006	F	0005
Jeffson creek	00011	670000	12/01/2014	00007	T	0006
Gainesville	00010	780000	11/30/2015	00008	F	0007
Orlando	00009	890000	03/12/2017	00009	F	0008
Orlando	00008	1000000	02/23/2012	00010	F	0009
Orlando	00007	340000	04/11/2014	00011	F	0009
Jacksonville	00006	450000	01/21/2015	00012	F	0010
Jacksonville	00004	560000	10/23/2015	00013	T	0011
High land park	00002	670000	08/01/2014	00014	T	0012
South lake	00001	110000	09/05/2015	00015	T	0001
South lake	00002	1010000	07/23/2014	00016	T	0002

3.1.1.11 Sales

INSERTION OF TABLE Sales (Example)

INSERT INTO Sales VALUES('000-00-0000');

Table 3.11 shows the states of Sales database schema.

Employee_id
000-00-0000
111-11-1111
222-22-2222
333-33-3333
444-44-4444
555-55-5555

3.1.1.12 Shift

INSERTION OF TABLE Shift (Example)

INSERT INTO Shift VALUES('000-00-0000', 'Mon');

Table 3.12 shows the states of Shift database schema.

Employee_id	Shift
000-00-0000	Mon

000-00-0000	Tue
111-11-1111	Wed
111-11-1111	Thu
222-22-2222	Fri
222-22-2222	Sat
222-22-2222	Sun
222-22-2222	Mon
333-33-3333	Tue
333-33-3333	Wed
444-44-4444	Thu
444-44-4444	Fri
555-55-5555	Sat
555-55-5555	Sun
555-55-5555	Tue

3.1.1.13 Manager

INSERTION OF TABLE Manager (Example)

INSERT INTO Manager VALUES (‘000-00-0000’, ‘10/10/2010’, ‘Spring Valley’);

Table 3.13 shows the states of Manager database schema.

Employee_id	Start_date	Sub_name
000-00-0000	10/10/2010	Spring Valley
111-11-1111	11/11/2010	Richardson

222-22-2222	12/21/2011	Dallas
333-33-3333	10/23/2012	Irvine
444-44-4444	01/11/2013	Jeffson creek
555-55-5555	12/01/2014	Gainesville
666-66-6666	11/30/2010	Orlando
777-77-7777	03/12/2011	Jacksonville
888-88-8888	02/23/2012	High land park
999-99-9999	04/11/2010	South lake

3.1.1.14 Secretary

INSERTION OF TABLE Secretary (Example)

INSERT INTO Secretary VALUES (‘666-66-6666’, 30);

Table 3.14 shows the states of Secretary database schema.

Employee_id	Hourly_range
666-66-6666	30
777-77-7777	40
888-88-8888	50
999-99-9999	60
101-01-0101	70
121-21-2121	80
131-31-3131	100

141-41-4141	30

3.1.1.15 Contract

INSERTION OF TABLE Contract (Example)

INSERT INTO Contract VALUES (‘C0001’, 200000, ‘10/10/2015’, ‘000-00-0000’);

Table 3.15 shows the states of Contract database schema.

Contract_id	Sale_price	Date_signed	Employee_id
C0001	200000	10/10/2015	000-00-0000
C0002	300000	11/12/2015	111-11-1111
C0003	400000	12/22/2014	222-22-2222
C0004	500000	10/23/2015	333-33-3333
C0005	600000	01/13/2014	444-44-4444
C0006	350000	12/01/2014	555-55-5555
C0007	450000	11/30/2015	666-66-6666
C0008	550000	03/12/2014	777-77-7777
C0009	400000	02/23/2015	888-88-8888
C0010	500000	04/11/2015	141-41-4141
C0011	350000	04/11/2015	555-55-5555

3.1.1.16 Associate

INSERTION OF TABLE Associate (Example)

INSERT INTO Associate VALUES (‘C0001’, ‘Spring Valley’, 0001, 00001);

Table 3.16 shows the states of Associate database schema.

Contract_id	sub_name	Floor_num	Lot_num
C0001	Spring Valley	0001	00001
C0002	Spring Valley	0001	00002
C0003	Richardson	0002	00001
C0004	Dallas	0001	00004
C0005	Dallas	0002	00005
C0006	Irvine	0005	00006
C0007	Jeffson creek	0006	00007
C0008	Gainesville	0007	00008
C0009	Orlando	0008	00009
C0010	Orlando	0009	00010
C0011	Orlando	0009	00011

3.1.1.17 Visit

INSERTION OF TABLE Visit (Example)

INSERT INTO Visit VALUES (‘jbutt@gmail.com’, ‘Spring Valley’, ‘10/10/2015’);

Table 3.17 shows the states of Visit database schema.

Customer_email	Sub_name	Visit_date
jbutt@gmail.com	Spring Valley	10/10/2015
josephine_darakjy@darakjy.org	Richardson	11/11/2015
art@venere.org	Dallas	12/21/2011
lpaprocki@hotmail.com	Irvine	10/23/2012
lpaprocki@hotmail.com	Richardson	01/11/2013
donette.foller@cox.net	Jeffson creek	12/01/2014
simona@morasca.com	Gainesville	11/30/2014
mitsue_tollner@yahoo.com	Orlando	03/12/2015
leota@hotmail.com	Jacksonville	08/23/2015
leota@hotmail.com	Spring Valley	08/11/2015
sage_wieser@cox.net	Richardson	01/21/2011
kris@gmail.com	Dallas	10/23/2012
minna_amigon@yahoo.com	Irvine	08/01/2010
amaclead@gmail.com	Spring Valley	09/05/2015
amaclead@gmail.com	Richardson	08/23/2015
kiley.caldarera@aol.com	Dallas	02/23/2014
gruta@cox.net	Irvine	05/11/2014
calbares@gmail.com	Richardson	01/21/2013
mattie@aol.com	Jeffson creek	10/13/2014
meaghan@hotmail.com	Gainesville	08/01/2014
gladys.rim@rim.org	Orlando	09/15/2013
yuki_whobrey@aol.com	Jacksonville	07/23/2013

3.1.1.18 Sign

INSERTION OF TABLE Sign (Example)

INSERT INTO Sign VALUES (‘C0001’, ‘yuki_whobrey@aol.com’, ‘Spring Valley’);

Table 3.18 shows the states of Sign database schema.

Contract_id	Customer_email	Sub_name
C0001	yuki_whobrey@aol.com	Spring Valley
C0002	mitsue_tollner@yahoo.com	Richardson
C0003	meaghan@hotmail.com	Dallas
C0004	mattie@aol.com	Irvine
C0005	leota@hotmail.com	Jeffson creek
C0006	kris@gmail.com	Gainesville
C0007	kiley.caldarera@aol.com	Orlando
C0008	gruta@cox.net	Jacksonville
C0009	donette.foller@cox.net	High land park
C0010	art@venere.org	High land park
C0011	art@venere.org	Gainesville

3.1.1.19 Tour

INSERTION OF TABLE Tour (Example)

INSERT INTO Tour VALUES (‘333-33-3333’, ‘jbutt@gmail.com’, ‘Spring Valley’, ‘10/10/2010’);

Table 3.19 shows the states of Tour database schema.

Employee_id	Customer_email	Sub_name	Date
333-33-3333	jbutt@gmail.com	Spring Valley	10/10/2010
444-44-4444	josephine_darakjy@darakjy.org	Richardson	11/11/2010
555-55-5555	art@venere.org	Dallas	12/21/2011
666-66-6666	lpaprocki@hotmail.com	Irvine	10/23/2012
777-77-7777	donette.foller@cox.net	Jeffson creek	01/11/2013
888-88-8888	simona@morasca.com	Gainesville	12/01/2014
999-99-9999	mitsue_tollner@yahoo.com	Orlando	11/30/2010
101-01-0101	leota@hotmail.com	Jacksonville	03/12/2011
121-21-2121	sage_wieser@cox.net	High land park	02/23/2012
131-31-3131	kris@gmail.com	South lake	04/11/2010
555-55-5555	art@venere.org	Richardson	01/21/2011
888-88-8888	lpaprocki@hotmail.com	Irvine	10/23/2012
999-99-9999	josephine_darakjy@darakjy.org	Orlando	08/01/2010

Till now we finished the process of creating tables and database states.

5.2 Creation of Views (Answer for Question d/Phase III)

3.2.1 School rank

This view shows the school rank of each subdivision along with the average base price of floor plans the subdivision provides.

```
CREATE VIEW School_Rank (Sub_name, Sub_school_rank, Avg_floor_price) AS
SELECT S.Name, S.School_rank, ROUND (AVG(F.price))
FROM SUBDIVISION S, FLOOR_PLAN F
WHERE S.name = F.Sub_name
GROUP BY S.School_rank, S.Name
```

3.2.2 Promising customers

This view shows the name, age and email address of potential customers with credit score over 740.

```
CREATE VIEW Promising_Customer AS
SELECT First_name, Last_name, Email
FROM CUSTOMER C
WHERE C.Customer_type = 'potential' AND C.Credit_score >= 740
```

3.2.3 Inventory homes

This view shows the information of unsold inventory homes that have been completed or will be completed by the end of 2015 for each subdivision along with the manager name of the subdivision.

```
CREATE VIEW Inventory_homes AS
SELECT I.*, E. First_name AS Manager_first_name, E.Last_name AS Manager_last_name
```

```
FROM INVENTORY I
RIGHT JOIN MANAGER M
ON I.Sub_name = M.Sub_name
RIGHT JOIN EMPLOYEE E
ON M.Employee_ID = E.Employee_ID
WHERE to_date(I.Move_in_date, 'MM/DD/YYYY') <= to_date('12/31/2015', 'MM/DD/YYYY') and I.availability = 'T'
```

3.2.4 Large floor plans

This view shows the information of floor plans over 4000 square feet for each subdivision.

```
CREATE VIEW Large_floor_plans AS
SELECT F.*
FROM FLOOR_PLAN F
WHERE F.Square_footage >= 4000
```

3.2.5 Sales record

This view shows, for each subdivision, the name of each sales agent and the number of contracts she prepared in year 2014.

```
CREATE VIEW Sales_record (Sale_first_name, Sale_last_name, Sale_Sub_name, Contract_num) AS
SELECT E.FIRST_NAME, E.Last_name, E.SUB_NAME, COUNT(C.CONTRACT_ID)
FROM EMPLOYEE E, CONTRACT C
WHERE C.DATE_SIGNED LIKE '%2014%' and C.EMPLOYEE_ID = E.EMPLOYEE_ID
GROUP BY E.FIRST_NAME, E.Last_name, E.Sub_name, C.EMPLOYEE_ID
```

3.2.6 Deal discount

This view shows, for each subdivision, the information of the contracts signed for inventory homes with a discount in year 2015. Here contract with discount means that the sales price showed in the contract is lower than the list price of the inventory home.

```
CREATE VIEW Deal_discount AS
SELECT C.*, I.Price
FROM CONTRACT C, SIGN S, INVENTORY I
WHERE C.SALE_PRICE < I.PRICE AND C.CONTRACT_ID = S.CONTRACT_ID
AND S.SUB_NAME = I.SUB_NAME
AND C.DATE_SIGNED LIKE '%2015%'
```

5.3 Creation of SQL Queries (Answer for Question e/Phase III)

Now we give out the SQL Queries for each of 14 questions listed in Question e as follows:

3.3.1. Retrieve the school rank of each subdivision in decreasing order of the average base price of floor plans the subdivision provides.

```
SELECT Sub_name, Sub_School_rank
FROM SCHOOL_RANK
ORDER BY AVG_FLOOR_PRICE DESC
```

3.3.2. For each subdivision, retrieve the number of unsold inventory homes that have been completed or will be ready by the end of year 2015.

```
SELECT SUB_NAME, COUNT(Inventory_Id)
From INVENTORY_HOMES
```

```
GROUP BY SUB_NAME
```

3.3.3. For each subdivision, retrieve information of the sales agents who prepared all the contracts with floor plans over 4000 square feet signed in year 2015.

```
CREATE VIEW FLOOR4000_2015 AS
SELECT C2.CONTRACT_ID, C2.EMPLOYEE_ID, L.Sub_name
FROM CONTRACT C2, SIGN S, LARGE_FLOOR_PLANS L
WHERE L.Sub_name = S.Sub_name AND S.CONTRACT_ID =
C2.CONTRACT_ID and C2.Date_signed LIKE '%2015%'

SELECT distinct E.*
FROM EMPLOYEE E, FLOOR4000_2015 F1
WHERE E.EMPLOYEE_ID = F1.EMPLOYEE_ID AND F1.Sub_name IN
(SELECT F.Sub_name
From FLOOR4000_2015 F
GROUP BY F.Sub_name
HAVING COUNT(distinct F.Employee_ID) = 1)
```

3.3.4. For each subdivision, retrieve the information of the sales agent who prepared the highest number of contracts in year 2014.

```
SELECT S.SALE_FIRST_NAME, S.SALE_LAST_NAME,
MAX(S.CONTRACT_NUM) AS MAX_CONTRACT_NUM
FROM SALES_RECORD S
GROUP BY S.SALE_SUB_NAME, S.SALE_FIRST_NAME,
S.SALE_LAST_NAME
```

3.3.5. Retrieve the information of the inventory home that has not been sold for the longest time since its completion, along with the name of the manager who is in charge of that subdivision.

```
SELECT I.*
FROM INVENTORY_HOMES I
WHERE NOT EXISTS
(SELECT I1.Move_in_date
FROM INVENTORY_HOMES I1
WHERE to_date(I.Move_in_date, 'MM/DD/YYYY') > to_date(I1.Move_in_date, 'MM/DD/YYYY'))
```

3.3.6. Retrieve the information of potential customers who have been visiting subdivisions of the company for more than one month and the price range of the inventory homes they have toured lies between $300,000 to $400,000.

```
SELECT C.*
FROM CUSTOMER C
WHERE C.Customer_type = 'potential' and C.email IN
((SELECT V.Customer_email
FROM VISIT V
WHERE to_date(V.Visit_date, 'MM/DD/YYYY') < SYSDATE - 30)
INTERSECT
(SELECT T.Customer_email
FROM TOUR T
WHERE T.Sub_name IN
(SELECT I.Sub_name
FROM Inventory I
WHERE I.Price >= 300000 and I.Price <= 400000)))
```

3.3.7. For each subdivision, retrieve the information of each customer whose home loan has not been approved over one month after he/she signed his/her contract. modified: time period between credit approve and sign > 1 month

```
SELECT C.*
FROM CUSTOMER C, LOAN_APPLICANT L, CONTRACT CON, SIGN S
WHERE S.CONTRACT_ID = CON.CONTRACT_ID AND S.CUSTOMER_EMAIL
= C.EMAIL
AND C.LOAN_ID = L.LOAN_ID AND to_date(CON.date_signed, 'MM/DD/YYYY')
- to_date(L.A_D_DATE, 'MM/DD/YYYY') > 30
```

3.3.8. Retrieve the name, age and email address of potential customers with credit score over 740 who visited a subdivision in August this year but have not visited any subdivision since September 1st.

```
(SELECT P.First_name, P.Last_name, P.Email
FROM VISIT V, Promising_customer P
WHERE V.Visit_date LIKE '08%2015' and V.Customer_email = P.email)
MINUS
(SELECT P1.First_name, P1.Last_name, P1.Email
FROM VISIT V1, Promising_customer P1
WHERE V1.Visit_date LIKE '09%2015' and V1.Customer_email = P1.email)
```

3.3.9. Retrieve the average discount received by the customers who have signed a contract to purchase an inventory home.

```
SELECT AVG(D.price - D.Sale_price)
FROM Deal_discount D
```

3.3.10. Retrieve information of the potential customers whose house preferences can be matched by one or more inventory homes that will be ready to move in by the end of year 2015

```
/* The target sqare of customers is the maximum square footage.
/* The target school rank of customer is the minimum square rank.
SELECT C.*
```

```
FROM CUSTOMER C, INVENTORY_HOMES I, FLOOR_PLAN F, SUBDIVISION
S
WHERE C.NUM_OF_BATH = F. NUM_OF_BATH AND C.NUM_OF_BEDROOM
= F.NUM_OF_BEDROOM AND C.SCHOOL_RANK >= S.SCHOOL_RANK
AND C.SQUARE_FOOTAGE <= F.SQUARE_FOOTAGE AND
C.CUSTOMER_TYPE = 'potential' AND S.NAME = F.SUB_NAME
AND S.NAME = I.SUB_NAME AND I.SUB_NAME = F.SUB_NAME AND
I.FLOOR_NUM = F.FLOOR_NUM
```

6. Dependency Diagram

We now draw a dependency diagram for each table from Figure 1.1 as follows:

2.1 Visit

There are two attributes together in the left-hand side of the functional dependencies, which is the key of relational schema Visit, Customer_emai and Sub_name. Therefore, every other attribute of this relational schema is functionally dependent on Customer_emai and Sub_name. The dependency diagram is shown as Figure 2.2

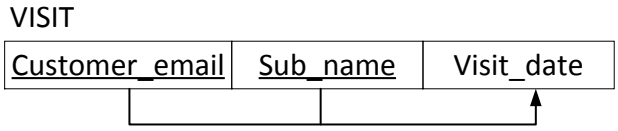

Figure 2.1. Dependency Diagram of Visit

2.2 Tour

There are three attributes together in the left-hand side of the functional dependencies, which is the key of relational schema Tour, Employee_id, Customer_email and Sub_name. Therefore, every other attribute of this relational

schema is functionally dependent on Employee_id, Customer_emai and Sub_name. The dependency diagram is shown as Figure 2.3

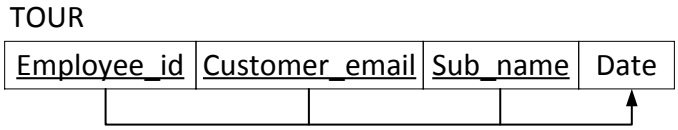

Figure 2.3. Dependency Diagram of Tour

2.3 Customer

There is only one attribute in the left-hand side of the functional dependencies, which is the key of relational schema Customer, Email. Therefore, every other attribute of this relational schema is functionally dependent on Email. The dependency diagram is shown as Figure 2.4

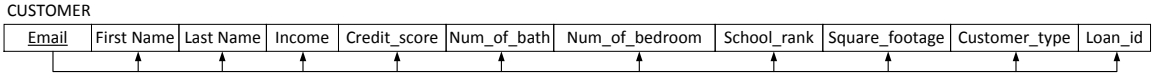

Figure 2.4. Dependency Diagram of Customer

2.4 Loan_applicant

There is only one attribute in the left-hand side of the functional dependencies, which is the key of relational schema Loan_applicant, Loan_id. Therefore, every other attribute of this relational schema is functionally dependent on Loan_id. The dependency diagram is shown as Figure 2.5

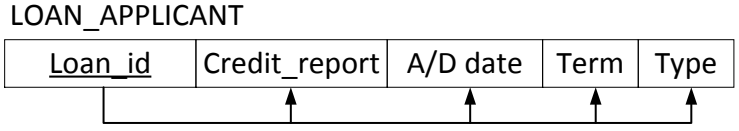

Figure 2.5. Dependency Diagram of Loan_applicant

2.5 Employee

There is only one attribute in the left-hand side of the functional dependencies, which is the key of relational schema Employee, Employee_id. Therefore, every other attribute of this relational schema is functionally dependent on Employee_id. The dependency diagram is shown as Figure 2.6

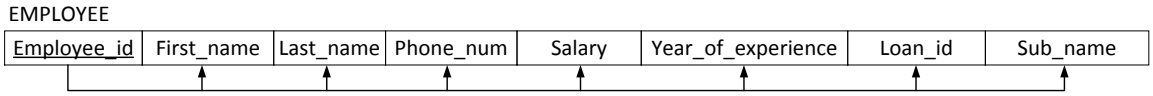

Figure 2.6. Dependency Diagram of Employee

2.6 Secretary

There is only one attribute in the left-hand side of the functional dependencies, which is the key of relational schema Secretary, Employee_id. Therefore, every other attribute of this relational schema is functionally dependent on Employee_id. The dependency diagram is shown as Figure 2.7

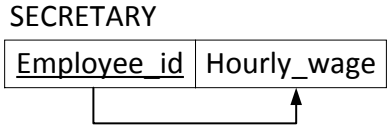

Figure 2.7. Dependency Diagram of Secretary

2.7 Manager

There is only one attribute in the left-hand side of the functional dependencies, which is the key of relational schema Manager, Employee_id. Therefore, every other attribute of this relational schema is functionally dependent on Employee_id. The dependency diagram is shown as Figure 2.8

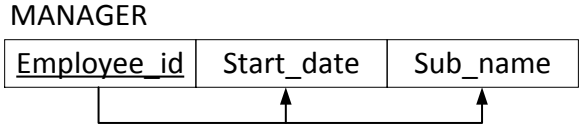

Figure 2.8. Dependency Diagram of Manager

2.8 Contract

There is only one attribute in the left-hand side of the functional dependencies, which is the key of relational schema Contract, Contract_id. Therefore, every other attribute of this relational schema is functionally dependent on Contract_id. The dependency diagram is shown as Figure 2.9

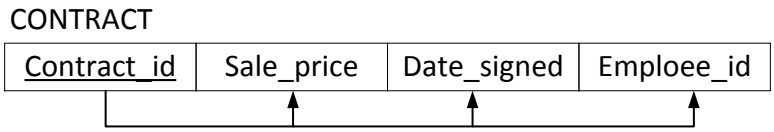

Figure 2.9. Dependency Diagram of Contract

2.9 Subdivision

There is only one attribute in the left-hand side of the functional dependencies, which is the key of relational schema Subdivision, Name. Therefore, every other attribute of this relational schema is functionally dependent on Name. The dependency diagram is shown as Figure 2.10.

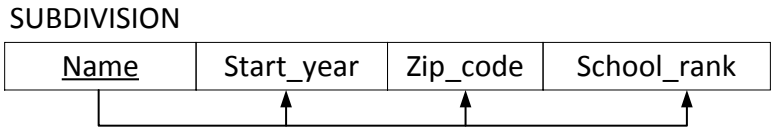

Figure 2.10. Dependency Diagram of Subdivision

2.10 Floor_plan

There are two attribute together in the left-hand side of the functional dependencies, which is the key of relational schema Floor_plan, Sub_name and Floor_num. Therefore, every other attribute of this relational schema is functionally dependent on Sub_name and Floor_num. The dependency diagram is shown as Figure 2.11

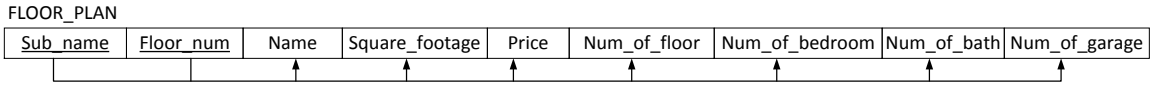

Figure 2.11. Dependency Diagram of Floor_plan

2.11 Inventory

There are two attribute together in the left-hand side of the functional dependencies, which is the key of relational schema Inventory, Sub_name and Inventory_id. Therefore, every other attribute of this relational schema is functionally dependent on Sub_name and Inventory_id.. The dependency diagram is shown as Figure 2.12

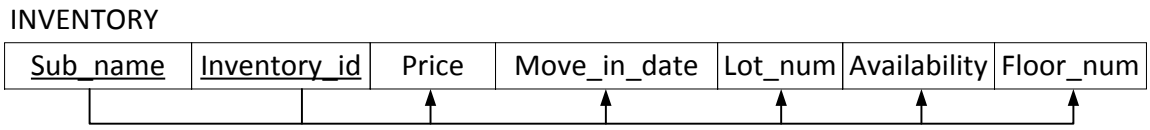

Figure 2.12. Dependency Diagram of Inventory

2.12 Lot

There are two attribute together in the left-hand side of the functional dependencies, which is the key of relational schema Lot, Sub_name and Lot_num. Therefore, every other attribute of this relational schema is functionally dependent on Sub_name and Lot_num. The dependency diagram is shown as Figure 2.13

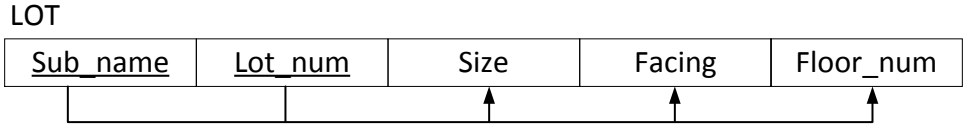

Figure 2.13. Dependency Diagram of Lot

2.13 Final Results

After drawing the dependency diagrams one after another, Figure 2.14 shows the final results for the whole database including the ones who do not have any

functional dependencies.

Figure 2.14. Whole Dependency Diagram for Home Builder Database

7. Conclusion

In this final report we summarized all the necessary descriptions and solutions for homebuilder company database, including process and result of EER diagrams, relational schemas in third normal form, SQL statements to create database, create view and solve corresponding queries, as well as dependency diagram. We also implement the whole database in Oracle and using a database state to test every query. In section 2 we also explained why we use superclass/subclass relationship to build relational schema, why we choose a Relational DBMS to implement our database, and the additional five business rules shown from implementation.