CPS510 A10
Project Report

Anna Fung 500567948
Hamza Iqbal 500973673
Abdulrehman Khan 500968727

Table of Contents

Introduction

Description of our Database:

In phase one of our database design project, we have chosen to create a Hotel Management Database Management System. The purpose of this database is to store persistent data to be required for our hotel's daily operations. It consists of entities which are: Employee, Rooms, Guest, Manager, Department, and Amenities. It also contains a relationship called 'Facilitates' which will form an entity/table itself due to the cardinality the relationship holds with other entities. The Database will contain functions such as booking a room or allowing the hotel staff to be contacted when necessary. These functions and their corresponding entities,attributes, etc will be discussed in further detail later on in this report. Lastly, more relationships between different entities will be explored.

Description of entities and their: application, functions, relationships and their information:

The entity Employee (Table 1) contains information about Employee ID, Names, Position, Department, Salary, Address, Phone Number, and email address. Employees provide different services depending upon the department they have been assigned. Another function is it allows the hotel to keep track of which employees are currently employed, this list should be updated when employees are hired or no longer work at the hotel. A third function is it allows for the hotel to send out emails or call staff when needed.

The entity Rooms will be decomposed into 2 tables: Room Table & Room Type table. The Room table will determine & identify the type of room that will be in use. The Room Type table will contain further information about the room at hand, such as floor, room number, price, type, cost per night, availability, and lastly which guest is occupying that room. One function is to keep track of the rooms to prevent rooms from getting double booked. It also allows customers to decide what rooms to book based on cost, room type, as well as accessibility, particularly if the room is very far from the main floor/lobby. Employees will be able to see which rooms are occupied, this can be helpful for housekeeping when cleaning rooms.

The entity Guest pertains to details regarding the guests staying at our hotel. This includes the type of room they are residing in, their room ID, personal information such as their Name, Contact Number, and Home Address. The main function of this entity is to ensure that we have a record available of guests currently staying at our hotel. Additionally, employees should have access and be able to update or add/remove guests based on check-in/check-out.

The entity Manager contains information regarding the various managers at our hotel. This includes their unique ID, Name, the Department they are responsible for, and their

individual salaries. The main functionality of this entity is that the manager has control of their specific department, and from there, their employee's salary, and the hiring of employees. This is possible as each employee is linked to a department ID, and each department has a manager.

The entity Department also contains 2 tables: Department & Department Type. The department table will be used to identify the department at hand. The Department Type table will be used to determine various attributes of the specific department, such as the type of department,yearly budget, & the number of employees. These are the various departments: Front Office, Housekeeping, Food & Beverage, Security department. Departments allow for structure and organization of employees and managers. The Department type also allows managers to keep track of the number of employees within each department, ensuring that each department is not under or overstaffed, and hence able to fund accordingly.

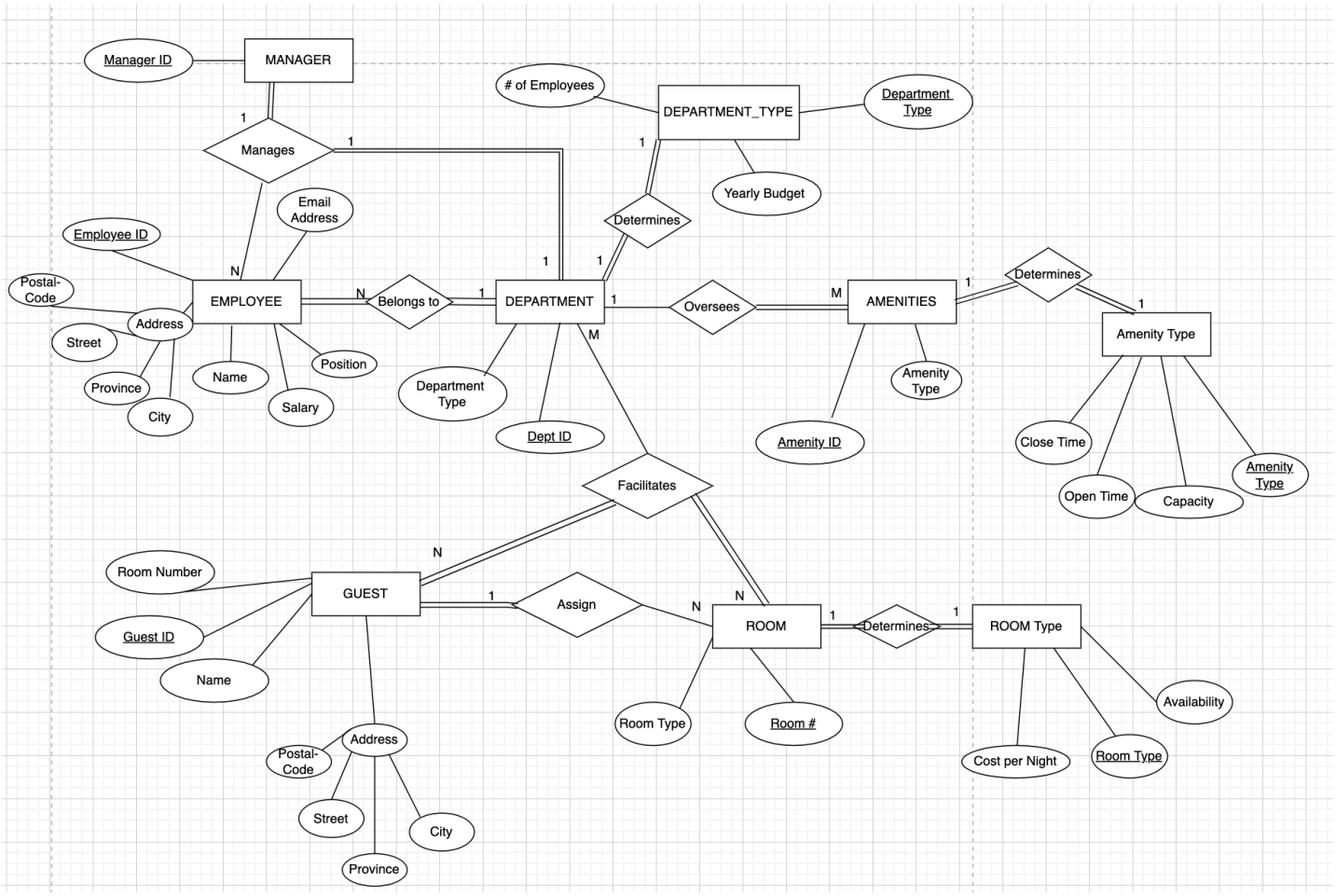
The entity Amenities allows for various services within the hotel. It also contains 2 further decomposed tables/entities, Amenities and Amenity Type. The Amenities table will be used to identify the Amenities at hand. The Amenity Type table will be used to determine various attributes of the specific Amenity, such as the type, open/close time, & capacity. It contains various amenities such as the gym, restaurant, pool & massage therapy.

The entity Facilitates is a relationship that exists between the department entity and the guest & room entities. Due to its M:N cardinality it will form an entity/table in itself, inheriting the keys from the entities that use the relationship. Its purpose is to allow the Department to accommodate guests and their rooms with various services.

Overview of Relationships:

Relationships exist between entities and work together to reduce redundancy. It also ensures that if a change is made, the change will be updated in all its related entities at the same time. One relationship is between guest and room, in which guests are assigned rooms to stay in. Another is between the employee and department, as employees are assigned to departments. A third relationship is between manager and department, where department is assigned to a manager. A fourth relationship is between employees and rooms, this assigns employees rooms to provide services for such as delivering food or cleaning rooms. One last relationship is between employees and managers, where managers are in charge of certain groups of employees depending on their department. Managers will be able to fire or hire employees, as well as give raises. Another relationship is the Facilitates relationship that exists between the department entity and the guest & room entities. Its role is to make sure the Department is able to accommodate and assist guests and their rooms.

Entity-Relation Diagram



Database Schema with Normalization & Functional Dependencies

MANAGER:

CREATE TABLE Manager (
  Manager_ID INTEGER PRIMARY KEY,
  Manager_DepartmentID INTEGER REFERENCES Department(Department_ID)
  );

Inserting Dummy Data:

  INSERT INTO Manager VALUES (001, 785);
  INSERT INTO Manager VALUES (002, 001);
  INSERT INTO Manager VALUES (003, 002);

```
INSERT INTO Manager VALUES (004, 003);
INSERT INTO Manager VALUES (005, 004);
```

Primary Key FD: {Manager_ID → Manager_DepartmentID }

Foreign Key FD: {Manager_DepartmentID → Manager_ID }

Since the relationship is 1:1, as for every department there exists only 1 manager, FD can be expressed both ways.

1NF: Table is in 1NF because all values are atomic.

2NF: Table is in 2NF because the table only contains 1 primary key (Manager_ID) and additionally all of the non-key attributes are fully functionally dependent on the primary key (Manager_ID).

3NF: The table is in 3NF because all of the non-key attributes are non-transitively dependent upon the primary key (Manager_ID).

BCNF Algorithm:

Let Manager_ID = A
Let Manager_DepartmentID = B

R(A,B)

The following are the FD's of the table above:

A → B
B → A

The following is the closure of the LHS:

$\{A\}^+ = \{A,B\}$ => A is a key
$\{B\}^+ = \{B,A\}$ => B is a key

Since A (Manager_ID) & B (Manager_DepartmentID) is on the LHS & is a key, this table is already in BCNF form.

	MANAGER_ID	MANAGER_DEPARTMENTID
1	1	785
2	2	1
3	4	3
4	5	4
5	3	2

Figure 1: Manager Table

EMPLOYEE:

```
CREATE TABLE Employee (
   Employee_ID                INTEGER PRIMARY KEY,
   Employee_Name              VARCHAR2(25) NOT NULL,
   Employee_EmailAdress       VARCHAR2(40) NOT NULL,
   Employee_Salary            INTEGER DEFAULT 0,
   Employee_DepartmentID      INTEGER  REFERENCES Department(Department_ID),
   Employee_AddressPostalCode VARCHAR(25),
   Employee_AddressStreet     VARCHAR(25),
   Employee_AddressProvince   VARCHAR(10),
   Employee_AddressCity       VARCHAR(25)
   );
```

Inserting Dummy Data:

```
   INSERT INTO Employee VALUES (001, 'John', 'john@gmail.com', 9999, 785, 'L2X 2S7', '999 victoria street', 'ON', 'Mississauga');
   INSERT INTO Employee VALUES (002, 'Daryl', 'daryl@gmail.com', 9999, 001, 'A2X 2W8', '999 desmond street', 'ON', 'Mississauga');
   INSERT INTO Employee VALUES (003, 'Billy', 'billy@gmail.com', 9999, 002, 'C2Z 1S0', '999 cadence street', 'ON', 'Mississauga');
   INSERT INTO Employee VALUES (004, 'Saunders', 'saunders@gmail.com', 9999, 003, 'Q2C 9F2', '999 phoenix street', 'ON', 'Mississauga');
   INSERT INTO Employee VALUES (005, 'Nozomi', 'nozomi@gmail.com', 9999, 004, 'A2O 3S9', '999 roche court', 'ON', 'Mississauga');
```

Primary Key FD: Employee_ID → Employee_Name, Employee_Salary, Employee_DepartmentID, Employee_AddressPostalCode, Employee_AddressStreet,Employee_AddressProvince, Employee_AddressCity

*NO Foreign Key FD due to N → 1 cardinality!
Every employee has a single, unique department ID.

1NF: Table is in 1NF because all values are atomic.

2NF: Table is in 2NF because the table only contains 1 primary key (Employee_ID) and additionally all of the non-key attributes are fully functionally dependent on the primary key (Employee_ID).

3NF: Table is in 3NF because all of the non-key attributes are non-transitively dependent upon the primary key (Employee_ID).

BCNF: Since the table is already in 3NF, this means that for BCNF form, A must be a candidate key given the FD A→ B .

Let A = Employee_ID
Let B = Employee_Name
Let C = Employee_Salary
Let D = Employee_DepartmentID
Let E = Employee_AddressPostalCode
Let F= Employee_AddressStreet
Let G= Employee_AddressProvince
Let H= Employee_AddressCity

Then, R(A,B,C,D,E,F,G,H)

Thus we have the FD:
A → B,C,D,E,F,G,H
Assume E → F,G,H

And thus the closure is:
A+ = {A,B,C,D,E,F,G,H} => A is the key
E+ = {E,F,G,H} => IS NOT A KEY so R is not in BCNF with respect to E → F,G,H

This must be decomposed into 2 tables R_1 & R_2:
R_1(A,B,C,D,E)
R_2(E,F,G,H) which is in BCNF

Is R_1 in BCNF with FD_1?

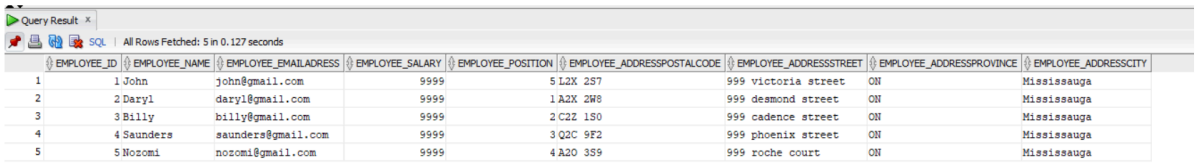
FD_1 = { A → B,C,D,E,F,G,H}

A+ = → B,C,D,E,F,G,H

FINAL BCNF schema for R is

R_1 = (A,B,C,D,E)
R_2 = (E,F,G,H)

Therefore we can conclude that this table is in BCNF form as for the FD A → B, C,D,E,F,G,H, A is both on the LHS and is the candidate key.



Query Result | SQL | All Rows Fetched: 5 in 0.127 seconds

	EMPLOYEE_ID	EMPLOYEE_NAME	EMPLOYEE_EMAILADRESS	EMPLOYEE_SALARY	EMPLOYEE_POSITION	EMPLOYEE_ADDRESSPOSTALCODE	EMPLOYEE_ADDRESSSTREET	EMPLOYEE_ADDRESSPROVINCE	EMPLOYEE_ADDRESSCITY
1	1	John	john@gmail.com	9999	5	L2X 2S7	999 victoria street	ON	Mississauga
2	2	Daryl	daryl@gmail.com	9999	1	A2X 2W8	999 desmond street	ON	Mississauga
3	3	Billy	billy@gmail.com	9999	2	C2Z 1S0	999 cadence street	ON	Mississauga
4	4	Saunders	saunders@gmail.com	9999	3	Q2C 9F2	999 phoenix street	ON	Mississauga
5	5	Nozomi	nozomi@gmail.com	9999	4	A2O 3S9	999 roche court	ON	Mississauga

Figure 2 : Employee Table

DEPARTMENT:

```
CREATE TABLE Department (
   Department_ID                    INTEGER PRIMARY KEY,
   Department_Type                  VARCHAR2(25) NOT NULL,
   Department_YearlyBudget          INTEGER DEFAULT 0,
   Department_NumberOfEmployees     INTEGER DEFAULT 0,

 );
```

Inserting Dummy Data:

```
   INSERT INTO Department VALUES (785,'HR',10000,5);
   INSERT INTO Department VALUES (001,'Front Office',9999,50);
   INSERT INTO Department VALUES (002,'House Keeping',9999,50);
   INSERT INTO Department VALUES (003,'Food and Beverage',9999,50);
   INSERT INTO Department VALUES (004,'Security',9999,50);
```

Primary Key FD: { Department_ID → Department_Type
Department_ID → Department_yearlyBudget
Department_ID → Department_NumberOfEmployees }

*NO Foreign Key in this table!

Primary Key FD: { Department_ID → Department_Type, Department_yearlyBudget, Department_NumberOfEmployees }

1NF: Table is in 1NF because all values are atomic.

2NF: Table is in 2NF because the table only contains 1 primary key (Department_ID) and additionally all of the non-key attributes are fully functionally dependent on the primary key (Department_ID).

3NF: Table was not in 3NF because all of the non-key attributes were not non-transitively dependent upon the primary key (Department_ID).

{ Department_Type → Department_YearlyBudget }
{ Department_Type → NumberOfEmployees }

Since this transitive dependency (TD) exists (as Department_Type is non-key), in order to fulfill this table as in 3NF it must be broken down into new tables to remove this TD.

NEW TABLES :

```
CREATE TABLE Department (
   Department_ID                    INTEGER PRIMARY KEY,
   Department_Type                  VARCHAR2(25) REFERENCES
Department_Type (Department_Type),
 );
```

DEPARTMENT_ID	DEPARTMENT_TYPE
785	HR
1	Front Office
2	House Keeping
3	Food and Beverage
4	Security
5	HR

Figure 3: Department Table

```
CREATE TABLE Department_Type (
   Department_Type                  VARCHAR2(25) PRIMARY KEY,
   Department_YearlyBudget          INTEGER DEFAULT 0,
   Department_NumberOfEmployees     INTEGER DEFAULT 0
 );
```

DEPARTMENT_TYPE	DEPARTMENT_YEARLYBUDGET	DEPARTMENT_NUMBEROFEMPLOYEES
HR	10000	5
Front Office	9999	50
House Keeping	9999	50
Food and Beverage	9999	50
Security	9999	50
HR	10000	5

Figure 4: Department_Type Table

BCNF Algorithm for Department Table:

Let Department_ID = A
Let Department_Type = B

R(A,B)

The following are the FD's of the table above:

A → B
B → A

The following is the closure of the LHS:

$\{A\}^+ = \{A,B\}$ => A is a key
$\{B\}^+ = \{B,A\}$ => B is a key

Since A (Department_ID), B (Department_Type) are both on the LHS & is a key, this table is already in BCNF form.

```
CREATE TABLE Department_Type (
   Department_Type                    VARCHAR2(25) PRIMARY KEY,
   Department_YearlyBudget            INTEGER DEFAULT 0,
   Department_NumberOfEmployees       INTEGER DEFAULT 0
 );
```

BCNF Algorithm for Department_Type Table:

Let Department_Type = A
Let Department_YearlyBudget = B
Let Department_NumberOfEmployees = C

R(A,B,C)

The following are the FD's of the table above:

A → B, C
AB → C
AC → B

The following is the closure of the LHS

$\{A\}^+$ = {A,B,C} => A is a key
{AB}+ = {A,B,C} => AB is key
{AC}+ = {A,C,B} => AC is key

Since A (Department_Type) is on the LHS & is a key, this table is already in BCNF form.

AMENITIES:

```
CREATE TABLE Amenities (
   Amenity_ID                          INTEGER PRIMARY KEY,
   Amenity_Type                        VARCHAR2 (25),
   Amenity_Capacity                    INTEGER ,
   Amenity_OpenTime                    VARCHAR(25) NOT NULL,
   Amenity_CloseTime                   VARCHAR(25) NOT NULL,
   Amenity_DepartmentID      INTEGER   REFERENCES Department(Department_ID)
   );
```

Inserting Dummy Data:

```
INSERT INTO Amenities VALUES (001, 'GYM', 20, '7:00 AM', '9:00 PM', 001);
INSERT INTO Amenities VALUES (002, 'RESTAURANT', 50, '7:00 AM', '9:00 PM', 003);
INSERT INTO Amenities VALUES (003, 'POOL', 20, '7:00 AM', '9:00 PM', 002);
INSERT INTO Amenities VALUES (004, 'MASSAGE THERAPY', 20, '7:00 AM', '9:00 PM', 001);
```

Primary Key FD: { Amenity_ID → Amenity_Type, Amenity_Capacity, Amenity_OpenTime, Amenity_CloseTime, Amenity_DepartmentID }

*NO Foreign Key FD due to M → 1 cardinality!

Amenity_DepartmentID → Amenity_ID

1NF: Table is in 1NF because all values are atomic.

2NF: Table is in 2NF because the table only contains 1 primary key (Amenity_ID) and additionally all of the non-key attributes are fully functionally dependent on the primary key (Amenity_ID).

3NF: Table is not in 3NF because all of the non-key attributes are not non-transitively dependent upon the primary key (Amenity_ID).

{ Amenity_Type → Amenity_Capacity, Amenity_OpenTime, Amenity_CloseTime }

Since this transitive dependency (TD) exists (as Amenity_Type is non-key), in order to fulfill this table as in 3NF it must be broken down into new tables to remove this TD.

NEW TABLES:

```
CREATE TABLE Amenities (
   Amenity_ID                                    INTEGER PRIMARY KEY,
   Amenity_Type          VARCHAR2 (25) REFERENCES Amenity_type (Amenity_type),
 Amenity_DepartmentID                  INTEGER REFERENCES Department(Department_ID)

   );
```

AMENITY_ID	AMENITY_TYPE	AMENITY_DEPARTMENTID
1	GYM	1
2	RESTAURANT	3
3	POOL	2
4	MASSAGE THERAPY	1

Figure 5: Amenities Table

```
CREATE TABLE Amenity_Type (
    Amenity_Type                                  VARCHAR2 (25) PRIMARY KEY,
   Amenity_Capacity                             INTEGER NOT NULL,
   Amenity_OpenTime                          VARCHAR(25) NOT NULL,
   Amenity_CloseTime                         VARCHAR(25) NOT NULL,
    );
```

AMENITY_TYPE	AMENITY_CAPACITY	AMENITY_OPENTIME	AMENITY_CLOSETIME
GYM	20	7:00 AM	9:00 PM
RESTAURANT	50	7:00 AM	9:00 PM
POOL	20	7:00 AM	9:00 PM
MASSAGE THERAPY	20	7:00 AM	9:00 PM

Figure 6: Amenity_Type Table

BCNF Algorithm:

Let Amenity_ID = A
Let Amenity_Type = B
Let Amenity_DepartmentID = C

R(A,B,C)

The following are the FD's of the table above:

A → B, C, D

The following is the closure of the LHS:

$\{A\}^+ = \{A,B,C,D\}$ => A is a key

Since A (Amenity_Type) is on the LHS & is a key, this table is already in BCNF form.

```
CREATE TABLE Amenity_Type (
    Amenity_Type                    VARCHAR2 (25) PRIMARY KEY,
   Amenity_Capacity                INTEGER NOT NULL,
   Amenity_OpenTime                VARCHAR(25) NOT NULL,
   Amenity_CloseTime               VARCHAR(25) NOT NULL,
    );
```

BCNF Algorithm:

Let Amenity_Type = A
Let Amenity_Capacity = B
Let Amenity_OpenTime = C
Let Amenity_CloseTime = D

R(A,B,C,D)

The following are the FD's of the table above:

A → B, C, D

The following is the closure of the LHS

$\{A\}^+ = \{A,B,C,D\}$ => A is a key

Since A (Amenity_Type) is on the LHS & is a key, this table is already in BCNF form.

GUEST:

```
CREATE TABLE Guest(
Guest_ID                    INTEGER PRIMARY KEY,
Guest_RoomNumber            INTEGER,
Guest_Name                  VARCHAR (25) NOT NULL,
Guest_AddressPostalCode     VARCHAR(25) NOT NULL,
Guest_AddressStreet         VARCHAR(25) NOT NULL,
Guest_AddressProvince       VARCHAR(10) NOT NULL,
Guest_AddressCity           VARCHAR(25) NOT NULL
);
```

Inserting Dummy Data:

```
INSERT INTO Guest VALUES (001, 123, 'Jerry','L2X 2S7', '999 isabelle street', 'ON',
'Mississauga');
   INSERT INTO Guest VALUES (002, 224, 'Springer','K9Z 4M0', '999 majed street', 'ON',
'Mississauga');
   INSERT INTO Guest VALUES (003, 335, 'Vadim', 'O9A 9AN', '999 abhari street', 'ON',
'Mississauga');
   INSERT INTO Guest VALUES (004, 462, 'Guerkov','W1N B2M', '999 lev street', 'ON',
'Mississauga');
   INSERT INTO Guest VALUES (005, 567, 'Alalala','LOS 1BN', '999 lowkey street', 'ON',
'Mississauga');
```

Primary Key FD: { Guest_ID → Guest_RoomNumber, Guest_Name, Guest_AddressPostalCode, Guest_AddressStreet, Guest_AddressProvince, Guest_AddressCity }

*NO Foreign Key in this table!

1NF: Table is 1NF because all values are atomic.

2NF: Table is 2NF, its candidate which is also the primary key (Guest_ID) and all of its non-key attributes are functionally dependent on its primary key (Guest_ID).

3NF: The Table is in 3NF as all non-key attributes are non-transitively dependent on its candidate key Guest_ID.

BCNF Algorithm:

Let Guest_ID = A
Let Guest_Name = B
Let Guest_AddressPostalCode = C
Let Guest_AddressStreet = D
Let Guest_AddressProvince = E

Let Guest_AddressCity = F

R(A,B,C,D,E,F)

The following are the FD's of the table above:

A → B,C,D,E,F
AB → C,D,E,F
Assume C → D,E,F

A+ = {A,B,C,D,E,F} IS A KEY
AB+ = {A,B,C,D,E,F} IS A KEY
C+ = {C,D,E,F} IS NOT A KEY so R is not in BCNF with respect to C → D,E,F

This must be decomposed into 2 tables R_1 & R_2:
R_1(A,B,C)
R_2(C,D,E,F) which is in BCNF

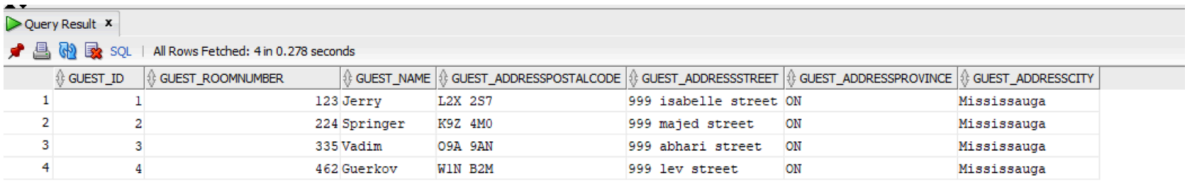
Is R_1 in BCNF with FD_1?

FD_1 = { A → B,C,D,E,F , AB → C,D,E,F}

A+ = {A,B,C,D,E,F} IS A KEY
AB+ = {C,D,E,F} IS A KEY

FINAL BCNF schema for R is

R_1 = (A,B,C)
R_2 = (C,D,E,F)


Query Result ×

SQL | All Rows Fetched: 4 in 0.278 seconds

	GUEST_ID	GUEST_ROOMNUMBER	GUEST_NAME	GUEST_ADDRESSPOSTALCODE	GUEST_ADDRESSSTREET	GUEST_ADDRESSPROVINCE	GUEST_ADDRESSCITY
1	1	123	Jerry	L2X 2S7	999 isabelle street	ON	Mississauga
2	2	224	Springer	K9Z 4M0	999 majed street	ON	Mississauga
3	3	335	Vadim	O9A 9AN	999 abhari street	ON	Mississauga
4	4	462	Guerkov	W1N B2M	999 lev street	ON	Mississauga

Figure 7: Guest Table

ROOM:

```
CREATE TABLE Room(
Room_RoomNumber                    INTEGER PRIMARY KEY,
Room_Type                          VARCHAR2 (20)  NOT NULL,
Room_CostPerNight                  INTEGER,
Room_Avaliability                  VARCHAR(1) NOT NULL,
Room_GuestID            INTEGER REFERENCES  Guest (Guest_ID),
Room_EmployeeID         INTEGER REFERENCES  Employee (Employee_ID)
);
```

Inserting Dummy Data:

```
INSERT INTO Room VALUES (001, 'Luxury',5000,'N', 001, 001);
INSERT INTO Room VALUES (002, 'Economic',5000,'N', 002, 001);
INSERT INTO Room VALUES (012, 'Luxury', 5000,'Y', 002, 001);
INSERT INTO Room VALUES (021, 'Standard',5000,'Y', 004, 002);
INSERT INTO Room VALUES (011, 'Luxury', 5000,'Y', 005, 002);
```

Primary Key FD: { Room_ID → Room_Number, Room_Type, Room_CostPerNight, Room_Avaliability, Room_GuestID, Room_EmployeeID }

*NO Foreign Key FD from Room_GuestID -> Room_ID due to N → 1 and no FD both ways for Room_EmployeeID & Room_ID since N → M cardinalities.

1NF: Table is 1NF because all values are atomic.

2NF: Table is 2NF, its candidate which is also the primary key (Room_RoomNumber) and all of its non-key attributes are functionally dependent on its primary key (Room_RoomNumber).

3NF: The table is not in 3NF as there is transitive dependency, to fix this decomposition is needed to bring it to 3NF form.

{ Room_Type → Room_CostPerNight, Room_Avaliability }

Since this transitive dependency (TD) exists (as Room_Type is non-key), in order to fulfill this table as 3NF it must be broken down into new tables to remove this TD.

New Tables:

```
CREATE TABLE Room(
   Room_RoomNumber                          INTEGER PRIMARY KEY,
   Room_Type          VARCHAR2(20) REFERENCES  Room_Type (Room_Type),
   Room_GuestID            INTEGER REFERENCES  Guest (Guest_ID),
   Room_EmployeeID         INTEGER REFERENCES  Employee (Employee_ID)
```

);

ROOM_ROOMNUMBER	ROOM_TYPE	ROOM_GUESTID	ROOM_EMPLOYEEID
1	Luxury	1	1
2	Economic	2	2
12	Luxury	3	3
21	Standard	4	4
11	Luxury	5	5

Figure 8: Room Table

```
CREATE TABLE  Room_Type(
   Room_Type                        VARCHAR2(20) PRIMARY KEY,
   Room_CostPerNight                INTEGER NOT NULL,
   Room_Avaliability                VARCHAR(1) NOT NULL,

);
```

ROOM_TYPE	ROOM_COSTPERNIGHT	ROOM_AVALIABILITY
Luxury	5000	N
Economic	5000	N
Luxury	5000	Y
Standard	5000	Y
Luxury	5000	Y

Figure 9: Room_Type Table

BCNF ALGORITHM:

Let Room_RoomNumber = A
Let Room_Type = B
Let Room_GuestID = C
Let Room_EmployeeID = D

R(A,B,C,D)

A → B,C,D

A+ = {A,B,C,D} IS A KEY, Therefore in BCNF.

```
CREATE TABLE  Room_Type(
   Room_Type                        VARCHAR2(20) PRIMARY KEY,
   Room_CostPerNight                INTEGER NOT NULL,
   Room_Avaliability                VARCHAR(1) NOT NULL,

);
```

BCNF ALGORITHM:

Let Room_Type = A
Let Room_CostPerNight = B
Let Room_Availability = C

R(A,B,C)

A → B,C

A+ = {A,B,C} IS A KEY, Therefore in BCNF.

FACILITATES:

```
CREATE TABLE FACILITATES(
   Department_ID       INTEGER REFERENCES Department(Department_ID),
   Guest_ID            INTEGER REFERENCES Guest(Guest_ID),
   Room_RoomNumber     INTEGER REFERENCES Room(Room_RoomNumber),
   PRIMARY KEY (Department_ID, Guest_ID, Room_RoomNumber)
   );
```

Inserting Dummy Data:

```
   INSERT INTO Facilitates VALUES (785,001,1);
   INSERT INTO Facilitates VALUES (1,002,2);
   INSERT INTO Facilitates VALUES (2,003,12);
   INSERT INTO Facilitates VALUES (3,004,21);
   INSERT INTO Facilitates VALUES (4,005,11);
```

*NO Primary and Foreign Key FD due to all M → N cardinalities!

	DEPARTMENT_ID	GUEST_ID	ROOM_ROOMNUMBER
1	1	2	2
2	2	3	12
3	3	4	21
4	4	5	11
5	785	1	1

Figure 10: Facilitates Table

Simple Database Queries (SQL & Relational Algebra)

SQL:

```
SELECT Manager_ID

FROM Manager

WHERE Manager_DepartmentID >= 3;
```

Relational Algebra:

$\pi_{Manager\_ID}(\sigma_{Manager\_DepartmentID >= 3}(Manager))$

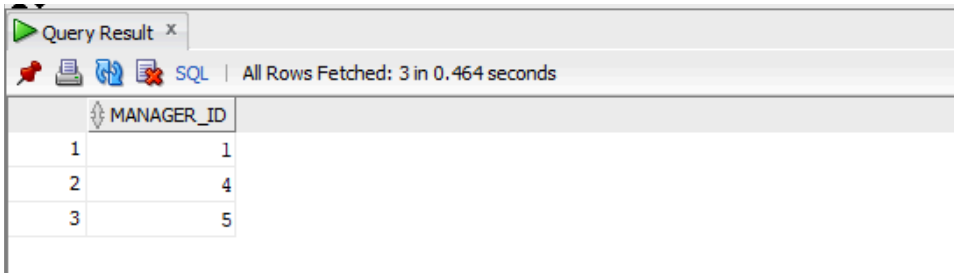


Figure 11: Example of Select Query for Manager Table

SQL:

```
SELECT Department_Type

FROM Department

WHERE Department_YearlyBudget >= 9999

ORDER BY Department_YearlyBudget DESC;
```

Relational Algebra:

$\pi_{Department\_Type}, \tau_{Department\_YearlyBudget\ DESC}(\sigma_{Department\_yearlyBudget >= 9999}(Department))$

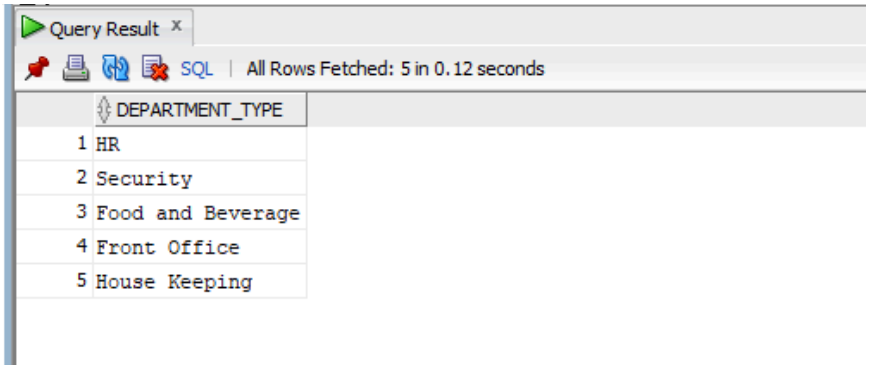


Figure 12: Example of Select and ORDER BY Query for Department Table

SQL:

```
SELECT Employee_ID
```

FROM Employee

WHERE Employee_AddressPostalCode = ‘Q2C 9F2’;

Relational Algebra:

$\pi_{Employee\_ID}(\sigma_{Employee\_AddressPostalCode = \text{‘Q2C 9F2’}}(Employee))$

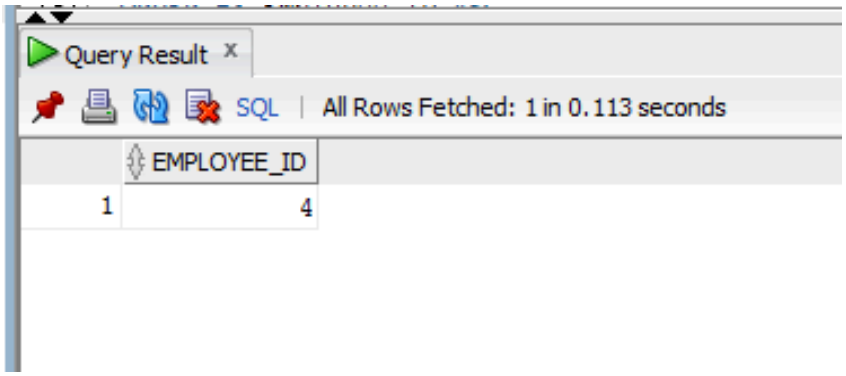


Figure 12: Example of Select Query for Employee Table

SQL:

SELECT Employee_ID, ' Employee email address is: ', Employee_EmailAdress

FROM Employee

WHERE Employee_ID > 000

ORDER BY Employee_ID ASC;

Relational Algebra:

$\pi_{Employee\_ID,\ \text{‘Employee email address is:’},\ Employee\_EmailAdress,\tau_{Employee\_ID\ ASC}}(\sigma_{Employee\_ID > 000}(Employee))$

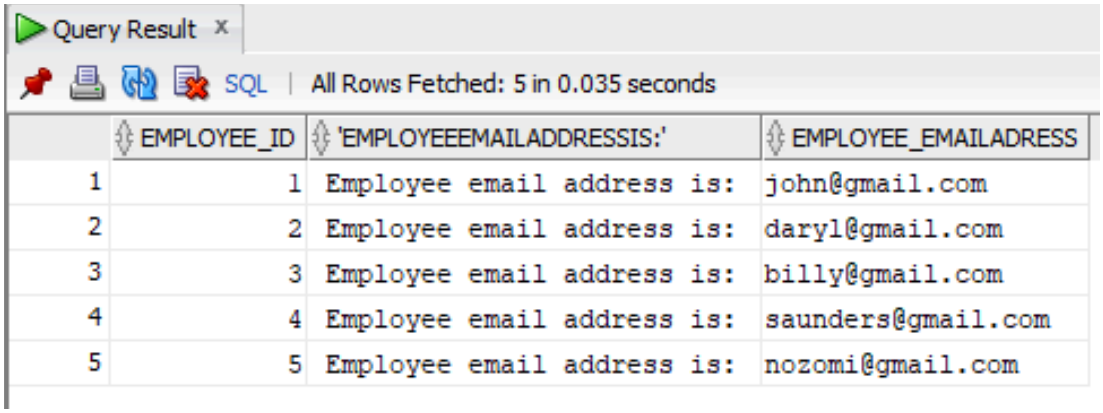


Figure 13: Example of Select and ORDER BY Query for Employee Table

SQL:

```
SELECT Amenity_ID, Amenity_Type

FROM Amenities

WHERE Amenity_Capacity = 50 ;
```

Relational Algebra:

$\pi_{\text{Amenity\_ID, Amenity\_Type}}(\sigma_{\text{Amenity\_Capacity = 50}}(\text{Amenities}))$

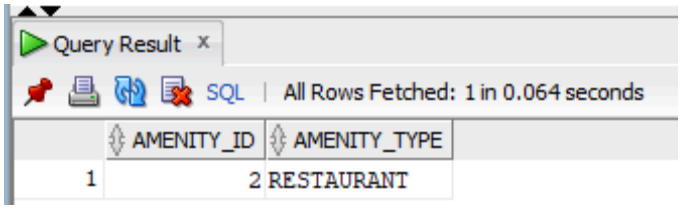


Figure 14: Example of Select Query for Amenity Table

SQL:

```
SELECT Guest_ID, Guest_Name

FROM Guest

WHERE Guest_RoomNumber = 224 OR Guest_RoomNumber = 462;
```

Relational Algebra:

$\pi_{\text{Guest\_ID, Guest\_Name}}(\sigma_{\text{(Guest\_RoomNumber = 224) OR (Guest\_RoomNumber = 462)}}(\text{Guest}))$

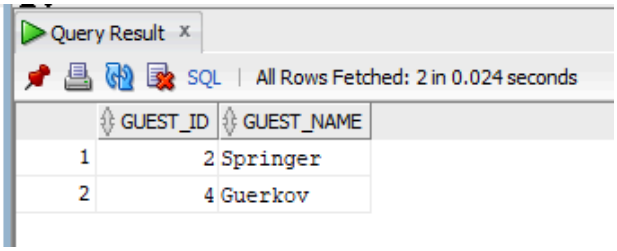


Figure 15: Example of Select Query for Guest Table

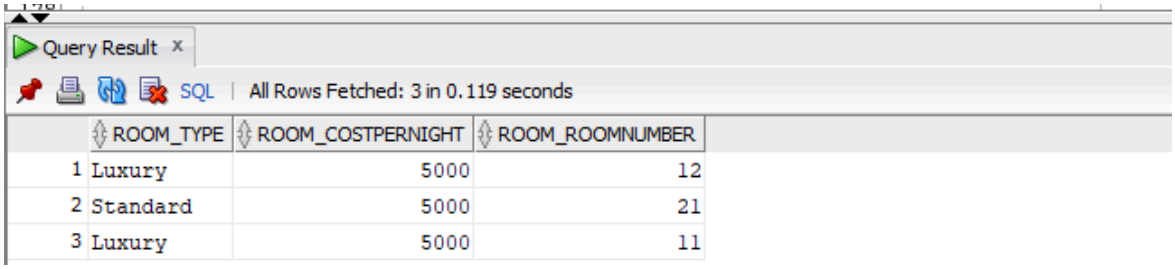
SQL:

```
SELECT Room_Type, Room_CostPerNight, Room_RoomNumber

FROM Room
```

WHERE Room_Avaliability = 'Y';

Relational Algebra:

$\pi_{Room\_Type,Room\_CostPerNight,Room\_RoomNumber}(\sigma_{Room\_Avalibility = 'Y'}(Room))$



	ROOM_TYPE	ROOM_COSTPERNIGHT	ROOM_ROOMNUMBER
1	Luxury	5000	12
2	Standard	5000	21
3	Luxury	5000	11

Figure 16: Example of Select Query for Room Table

SQL:

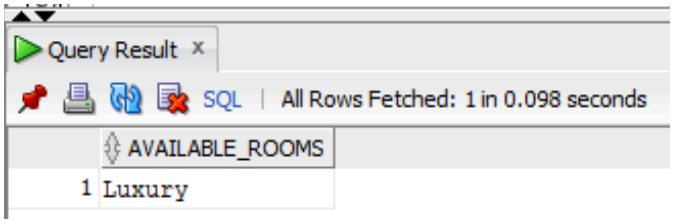
SELECT Distinct Room_Type AS Available_Rooms

FROM Room

WHERE Room_Avaliability = 'Y' AND Room_Type = 'Luxury';

Relational Algebra:

$\pi_{Room\_Type}\ \rho_{\ Available\_Rooms}\ \leftarrow_{\ Room\_Type}(\sigma_{Room\_Availabiility= 'Y' \text{ AND } Room\_Type = ''Luxury'}(Room))$



	AVAILABLE_ROOMS
1	Luxury

Figure 17: Example of Select Distinct Query for Room Table

SQL:

SELECT Department_NumberOfEmployees

FROM Department

GROUP BY Department_NumberOfEmployees;

Relational Algebra:

$\pi_{Department\_NumberOfEmpoyees}(\sigma_{Department\_NumberOfEmployees}(Department))$

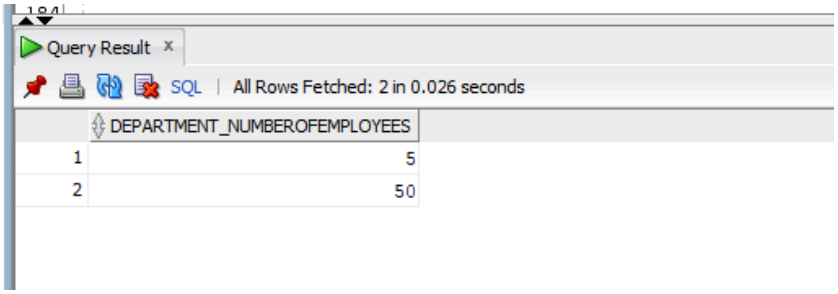


Figure 18: Example of Select and Group By Query for Department Table

Join & View Queries of Tables (SQL and Relational Algebra)

Joining Department Table and Manager Table:

SQL:

SELECT Department.Department_ID, Department_Type, Department_YearlyBudget, Department_NumberofEmployees

FROM Department

WHERE Department_NumberofEmployees >=50

ORDER BY Department.Department_ID DESC;

Relational Algebra:

π Department.Department_ID,Department_Type,Department_YearlyBudget,Department_NumberofEmployees,τDepartment.Department_ID ASC
$\sigma_{\text{Department\_NumberofEmployees >= 50}}$ (Department)

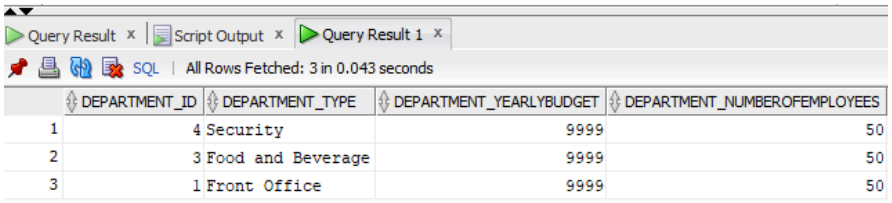


Figure 19: Joining Department and Manager Table Query

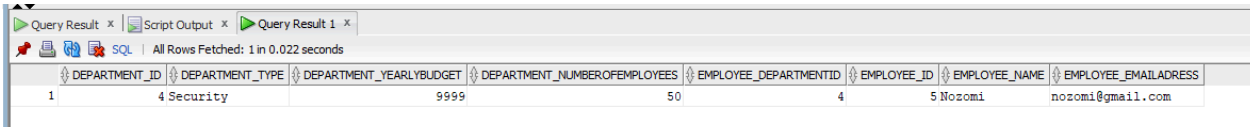
Joining Department Table and Employee Table:

SQL:

```
SELECT d.Department_ID, Department_Type, Department_YearlyBudget,
Department_NumberofEmployees, e.Employee_DepartmentID, Employee_ID,
Employee_Name, Employee_EmailAdress

FROM Department d, Employee e

WHERE Department_YearlyBudget >= 9999
    AND d.Department_ID >=1
    AND Department_Type = 'Security'
    AND e.Employee_DepartmentID = d.Department_ID


ORDER BY Department_YearlyBudget DESC;
```

Relational Algebra:

π d.Department_ID,Department_Type,Department_YearlyBudget,Department_NumberofEmployees,τDepartment.Department_ID ASC
σDepartment_NumberofEmployees >= 50 (Department)



	DEPARTMENT_ID	DEPARTMENT_TYPE	DEPARTMENT_YEARLYBUDGET	DEPARTMENT_NUMBEROFEMPLOYEES	EMPLOYEE_DEPARTMENTID	EMPLOYEE_ID	EMPLOYEE_NAME	EMPLOYEE_EMAILADRESS
1	4	Security	9999	50	4	5	Nozomi	nozomi@gmail.com

Figure 20: Joining Department and Employee Table Query

Joining Room Table and Guest Table:

SQL:

```
SELECT Room_Type, Room_CostPerNight, Guest_Name,Guest.Guest_ID,
Room.Room_GuestID,Guest.Guest_RoomNumber,
Room.Room_RoomNumber

FROM Room, Guest

WHERE Guest.Guest_ID = Room.Room_GuestID AND Room_Type='Luxury';
```

Relational Algebra:

πRoom_Type, Room_CostPerNight, Guest_Name, Guest.Guest_ID, Room.Room_GuestID, Guest.Guest_RoomNumber, Room.Room_RoomNumber
(σGuest.Guest_ID = Room.Room_GuestID AND Room_Type = 'Luxury' (Room >< Guest))

	ROOM_TYPE	ROOM_COSTPERNIGHT	GUEST_NAME	GUEST_ID	ROOM_GUESTID	GUEST_ROOMNUMBER	ROOM_ROOMNUMBER
1	Luxury	5000	Jerry	1	1	123	1
2	Luxury	5000	Springer	2	2	224	12

Figure 21: Joining Department and Guest Table Query

Joining Amenity Table and Department Table:

SQL:

```
SELECT Amenity_Type, Department_Type, Amenity_ID

FROM  Amenities, Department

WHERE Amenity_DepartmentID = Department_ID
AND  Amenity_OpenTime = '7:00 AM'  AND Amenity_Capacity <= 20;
```

Relational Algebra:

$\pi_{\text{Amenity\_Type, Department\_Type, Amenity\_ID}}$ ($\sigma_{\text{Amenity\_DepartmentID = Department\_ID AND Amenity\_OpenTime = '7:00 AM' AND Amenity\_Capacity <= 20}}$ (Amenities >< Department))

	AMENITY_TYPE	DEPARTMENT_TYPE	AMENITY_ID
1	GYM	Front Office	1
2	POOL	House Keeping	3
3	MASSAGE THERAPY	Front Office	4

Figure 22: Joining Amenity and Department Table Query

View for Guests Staying in Luxury Rooms:

SQL:

```
CREATE VIEW Guest_Luxury_Room AS

SELECT Room_Type, Room_CostPerNight,Guest_Name

FROM Room, Guest

WHERE Guest.Guest_ID = Room.Room_GuestID
AND Room_Type='Luxury';


SELECT *FROM Guest_Luxury_Room;
```

Relational Algebra:

$\pi_{Room\_Type,\ Room\_CostPerNight,\ Guest\_Name}\ (\sigma_{Guest\_ID=Room\_GuestID\ AND\ Room\_Type\ =\ 'Luxury'}\ (Room >< Guest))$

	ROOM_TYPE	ROOM_COSTPERNIGHT	GUEST_NAME
1	Luxury	5000	Jerry
2	Luxury	5000	Springer

Figure 23: View for Guests Residing in Luxury Rooms

View for Large Departments within hotel:

SQL:

```
CREATE VIEW Large_Departments AS

(SELECT *

FROM Department

WHERE Department_NumberOfEmployees >=50);

SELECT *FROM Large_Departments;
```

Relational Algebra:

$\sigma_{Department\_NumberOfEmployees>=\ 50}\ (Department)$

Query Result | Script Output | Query Result 1 | Query Result 2 | Query Result 3 | Query Result 4 | Query

SQL | All Rows Fetched: 4 in 0.021 seconds

	DEPARTMENT_ID	DEPARTMENT_TYPE	DEPARTMENT_YEARLYBUDGET	DEPARTMENT_NUMBEROFEMPLOYEES
1	1	Front Office	9999	50
2	2	House Keeping	9999	50
3	3	Food and Beverage	9999	50
4	4	Security	9999	50

Figure 24: View for Large Department within hotel

View for Economic Guests residing at hotel:

SQL:

```
CREATE VIEW Economic_Guests AS

(SELECT Room_Type, Guest_Name, Guest_RoomNumber, Room_CostPerNight
```

```
FROM Guest, Room

WHERE Room_Type = 'Economic'
AND NOT Room_Type = 'Luxury'
AND NOT Room_Type = 'Standard'
AND Guest.Guest_ID = Room.Room_GuestID);

SELECT *FROM Economic_Guests
```

Relational Algebra:

$\pi_{\text{Room\_Type, Guest\_Name, Guest\_RoomNumber, Room\_CostPerNight}}(\sigma_{\text{Room\_Type = 'Economic' AND NOT Room\_Type = 'Luxury' AND NOT Room\_Type = 'Standard' AND Guest.Guest\_ID = Room.Room\_GuestID}}(\text{Guest} >< \text{Room}))$

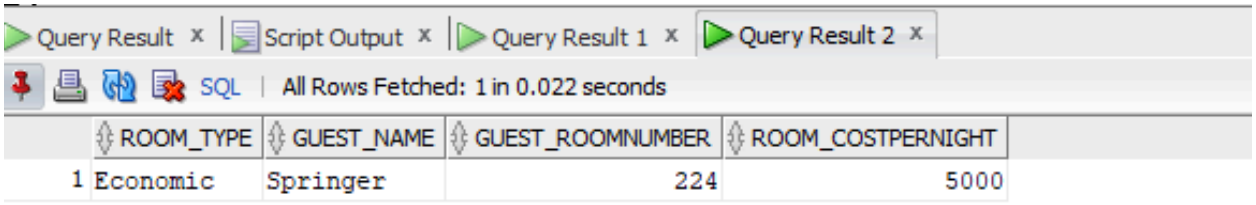


Figure 25: View for Economic Guests residing within hotel

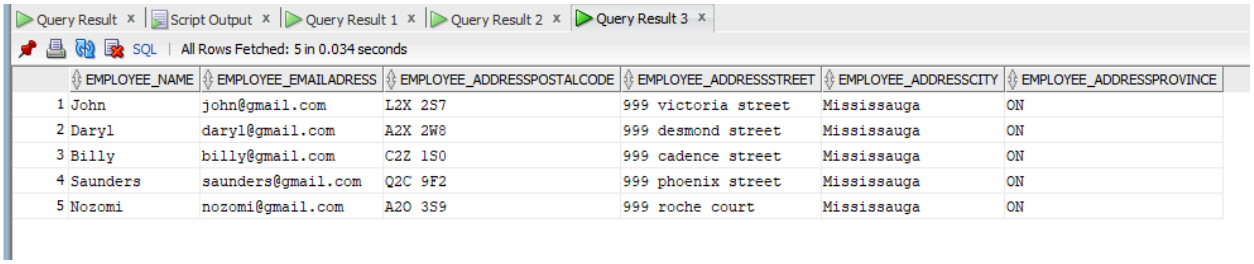
View for Employee Personal Information:

SQL:

```
CREATE VIEW Employee_PersonalInformation AS

(SELECT Employee_Name, Employee_EmailAdress, Employee_AddressPostalCode,
Employee_AddressStreet, Employee_AddressCity, Employee_AddressProvince

FROM Employee);

SELECT *FROM Employee_PersonalInformation
```

Relational Algebra:

$\pi_{\text{Employee\_Name, Employee\_EmailAdress, Employee\_AddressPostalCode, Employee\_AddressStreet, Employee\_AddressCity, Employee\_AddressProvince}}(\text{Employee})$



Query Result | Script Output | Query Result 1 | Query Result 2 | Query Result 3

SQL | All Rows Fetched: 5 in 0.034 seconds

	EMPLOYEE_NAME	EMPLOYEE_EMAILADRESS	EMPLOYEE_ADDRESSPOSTALCODE	EMPLOYEE_ADDRESSSTREET	EMPLOYEE_ADDRESSCITY	EMPLOYEE_ADDRESSPROVINCE
1	John	john@gmail.com	L2X 2S7	999 victoria street	Mississauga	ON
2	Daryl	daryl@gmail.com	A2X 2W8	999 desmond street	Mississauga	ON
3	Billy	billy@gmail.com	C2Z 1S0	999 cadence street	Mississauga	ON
4	Saunders	saunders@gmail.com	Q2C 9F2	999 phoenix street	Mississauga	ON
5	Nozomi	nozomi@gmail.com	A2O 3S9	999 roche court	Mississauga	ON

Figure 26: View for Employee Personal Information

Advanced Database Queries (SQL & Relational Algebra)

SQL:

SELECT 'Average Salary for an employee is: ', AVG(Employee_Salary)

FROM Employee;

Relational Algebra:

$\pi_{\text{AVG (Employee\_Salary)}}$ (Employee)

	'AVERAGESALARYFORANEMPLOYEEIS:'	AVG(EMPLOYEE_SALARY)
1	Average Salary for an employee is:	9999

Figure 27: Advanced query using AVG function

SQL:

SELECT Room_Type, Room_RoomNumber

FROM Room r

WHERE EXISTS

(SELECT Room_GuestID
FROM Room r1, Guest g1
WHERE r1.Room_GuestID = g1.Guest_ID);

Relational Algebra:

$\pi_{\text{Room\_Type, Room\_RoomNumber}}$ (σ_{EXISTS} ($\pi_{\text{Room\_GuestID}}$($\sigma_{\text{Room\_GuestID = Guest\_ID}}$ (Room >< Guest)))) (Room)

	ROOM_TYPE	ROOM_ROOMNUMBER
1	Luxury	1
2	Economic	2
3	Luxury	12
4	Standard	21
5	Luxury	11

Figure 28: Advanced query using nested Select and EXISTS function

SQL:

```
SELECT  Employee_Name
FROM Employee e, Department d
WHERE e.Employee_DepartmentID = d.Department_ID AND NOT EXISTS
(SELECT Department_ID
 FROM Department d
 WHERE d.Department_ID=577);
```

Relational Algebra:

$\pi_{\text{Employee\_Name}}(\pi_{\text{Department\_ID}}\sigma_{\text{Department\_ID = 577}}(\text{Department}))(\text{Employee} >< \text{Department})$

	EMPLOYEE_NAME
1	John
2	Daryl
3	Billy
4	Saunders
5	Nozomi

Figure 29: Advanced query using nested Select and NOT EXISTS function

```
SELECT COUNT(*) FROM (
SELECT Guest_AddressCity FROM Guest
WHERE Guest_AddressCity = 'Mississauga'
UNION ALL
SELECT Employee_AddressCity FROM Employee
WHERE Employee_AddressCity = 'Mississauga')
```

	COUNT(*)
1	10

SQL:

```
SELECT Department_YearlyBudget, AVG(Department_NumberOfEmployees)
FROM Department
```

```
GROUP BY Department_YearlyBudget

HAVING AVG(Department_NumberOfEmployees) < (SELECT
AVG(Department_NumberOfEmployees)

 FROM Department);
```

Relational Algebra:

$\pi_{\text{Department\_YearlyBudget, AVG (Department\_NumberOfEmployees)}} (\pi_{\text{Department\_ID}} \sigma_{\text{Department\_ID = 577}}(\text{Department}))(\text{Employee} >< \text{Department})$

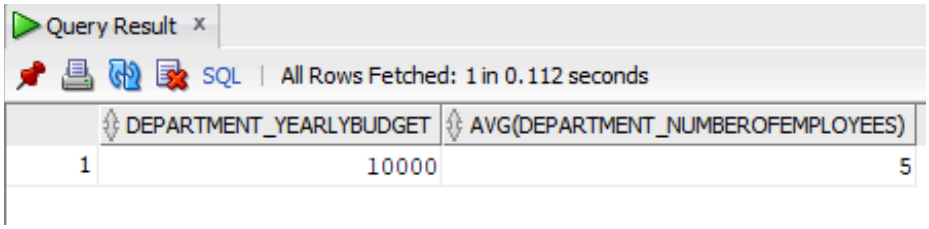


Figure 30: Advanced query using HAVING AVG and nested select with group by

SQL:

```
SELECT  MAX (d.Department_YearlyBudget), MIN (a.Amenity_Capacity), AVG
(e.Employee_Salary), VARIANCE (e.Employee_Salary), STDDEV (e.Employee_Salary)

FROM Department d, Amenities a, Employee e
```

Relational Algebra:

π MAX department_YearlyBudget, MIN Amenity_Capacity, AVG Employee_Salary, VARIANCE Employee_Salary, STDDEV Employee_Salary
(Department >< Amenities >< Employee)

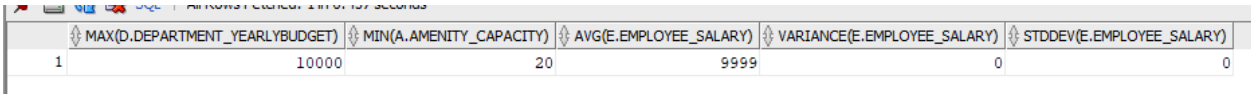


Figure 31: Advanced query using MAX, MIN, AVG, VARIANCE, and STDDEV function

Unix Shell Implementation

Menu:

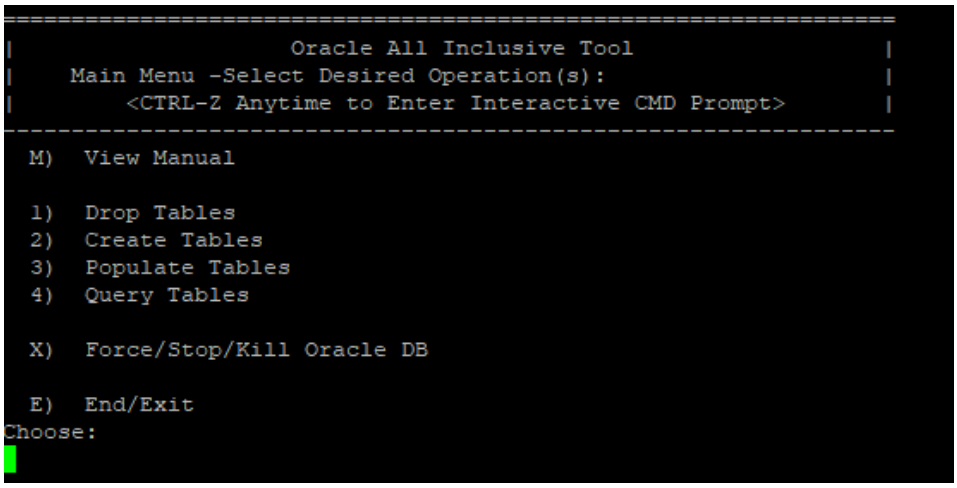


Figure 32: UNIX SHELL MENU

Dropping table:

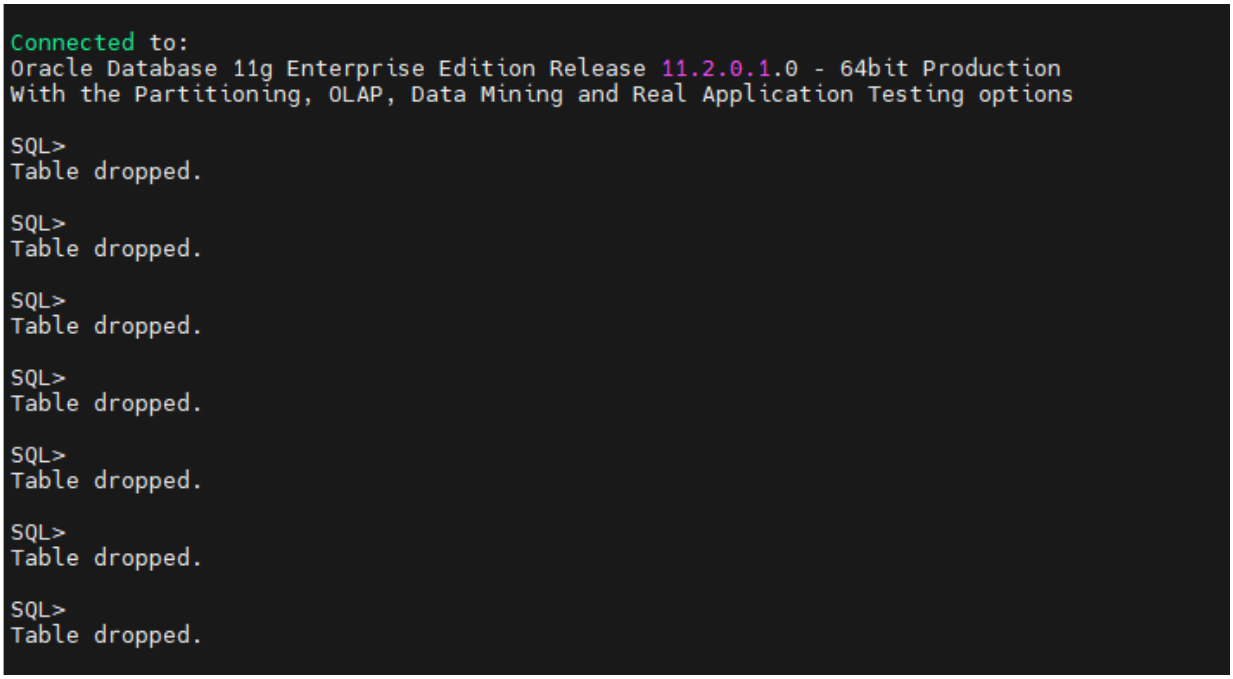


Figure 33: Dropping Tables within UNIX SHELL

Create tables:

```
Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> SQL>   2    3    4    5    6
Table created.

SQL> SQL>   2    3    4
Table created.

SQL> SQL>   2    3    4    5    6    7    8    9   10   11
Table created.

SQL> SQL> SQL>   2    3    4    5    6    7    8
Table created.

SQL> SQL>   2    3    4    5    6    7    8    9
Table created.

SQL> SQL>   2    3    4    5    6    7    8
Table created.

SQL> SQL>   2    3    4    5    6    7
Table created.

SQL> SQL> Disconnected from Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
a19khan@elara:~$
```

Figure 34: Creating Tables within UNIX SHELL

Populating tables

```
Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> SQL> SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL> SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL> SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL> SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.
```

Figure 35: Populating Tables within UNIX SHELL

Simple Queries

```
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> SQL> SQL> SQL>   2    3    4
DEPARTMENT_TYPE
-------------------------
HR
Security
Food and Beverage
Front Office
House Keeping

SQL> SQL>   2    3
MANAGER_ID
----------
         1
         4
         5

SQL> SQL>   2    3
ROOM_TYPE            ROOM_COSTPERNIGHT ROOM_ROOMNUMBER
-------------------- ----------------- ---------------
Luxury                            5000              12
Standard                          5000              21
Luxury                            5000              11

SQL> SQL>   2    3    4    5    6
ROOM_TYPE            ROOM_ROOMNUMBER
-------------------- ---------------
Luxury                             1
Economic                           2
Luxury                            12
Standard                          21
Luxury                            11

SQL> SQL> SQL> SQL> SQL> Disconnected from Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
```

Figure 36: Simple Queries within UNIX SHELL

Advanced Queries

```
Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> SQL>   2
'AVERAGESALARYFORANEMPLOYEEIS:'    AVG(EMPLOYEE_SALARY)
---------------------------------- --------------------
Average Salary for an employee is:                 9999

SQL> SQL> SQL>   2    3    4    5    6
ROOM_TYPE            ROOM_ROOMNUMBER
-------------------- ---------------
Luxury                             1
Economic                           2
Luxury                            12
Standard                          21
Luxury                            11

SQL> SQL> SQL>   2    3    4    5    6
EMPLOYEE_NAME
-------------------------
John
Daryl
Billy
Saunders
Nozomi

SQL> SQL> SQL>   2    3    4    5    6    7    8  SQL> SQL>   2    3    4    5
DEPARTMENT_YEARLYBUDGET AVG(DEPARTMENT_NUMBEROFEMPLOYEES)
----------------------- ----------------------------------
                  10000                                  5

SQL> SQL> SQL>   2    3  SQL> SQL> Disconnected from Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
a19khan@elara:~$
```

Figure 37: Advanced Queries within UNIX SHELL

Java UI

CODE:

```
package jdbcoracleconnectiontemplate;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.ResultSet;
import java.util.Properties;

public class JdbcOracleConnectionTemplate {

        static final String QUERY = "SELECT id, first, last, age FROM DUMMYTABLE";


 public static void main(String[] args) {

       Connection conn1 = null;

       try {
          Class.forName("oracle.jdbc.OracleDriver");


           String dbURL1 = "jdbc:oracle:thin:removed user+pass
@oracle.scs.ryerson.ca:1521:orcl";

                              conn1 = DriverManager.getConnection(dbURL1);
          if (conn1 != null) {
             System.out.println("Connected with connection #1");
          }

                              String query = "select NAME, NUM from TESTJDBC";

                              try (Statement stmt = conn1.createStatement()) {



                    String sq1 = "CREATE TABLE DUMMYTABLE " +
                               "(id INTEGER not NULL, " +
                               " first VARCHAR(255), " +
```

```
                " last VARCHAR(255), " +
                " age INTEGER, " +
                " PRIMARY KEY ( id ))";

        String sq2 = "CREATE TABLE TESTINGTABLE " +
                "(id INTEGER not NULL, " +
                " first VARCHAR(255), " +
                " last VARCHAR(255), " +
                " age INTEGER, " +
                " PRIMARY KEY ( id ))";

         stmt.executeUpdate(sq1);
         stmt.executeUpdate(sq2);



        String Dropsql = "DROP TABLE DUMMYTABLE";
        String Dropsq2 = "DROP TABLE TESTINGTABLE";
        stmt.executeUpdate(Dropsql);
        stmt.executeUpdate(Dropsq2);


         String sql = "INSERT INTO DUMMYTABLE VALUES(13, 'hey', 'no', 128)";
         stmt.executeUpdate(sql);



                ResultSet rs = stmt.executeQuery(query);

                //If everything was entered correctly, this loop should print each row of
data in your TESTJDBC table.
                // And you should see the results as follows:
                // Connected with connection #1
                // ALIS, 67
                // BOB, 345



                while (rs.next()) {

           System.out.print("ID: " + rs.getInt("id"));
           System.out.print(", Age: " + rs.getInt("age"));
```

```
                    System.out.print(", First: " + rs.getString("first"));
                    System.out.println(", Last: " + rs.getString("last"));

                            /*String name = rs.getString("NAME");
                            int num = rs.getInt("NUM");
                            System.out.println(name + ", " + num);*/
                    }
                    } catch (SQLException e) {
                            System.out.println(e.getErrorCode());
                    }



        } catch (ClassNotFoundException ex) {
            ex.printStackTrace();
        } catch (SQLException ex) {
            ex.printStackTrace();
        } finally {
            try {
                if (conn1 != null && !conn1.isClosed()) {
                    conn1.close();
                }

            } catch (SQLException ex) {
                ex.printStackTrace();
            }
        }


    }
}
```

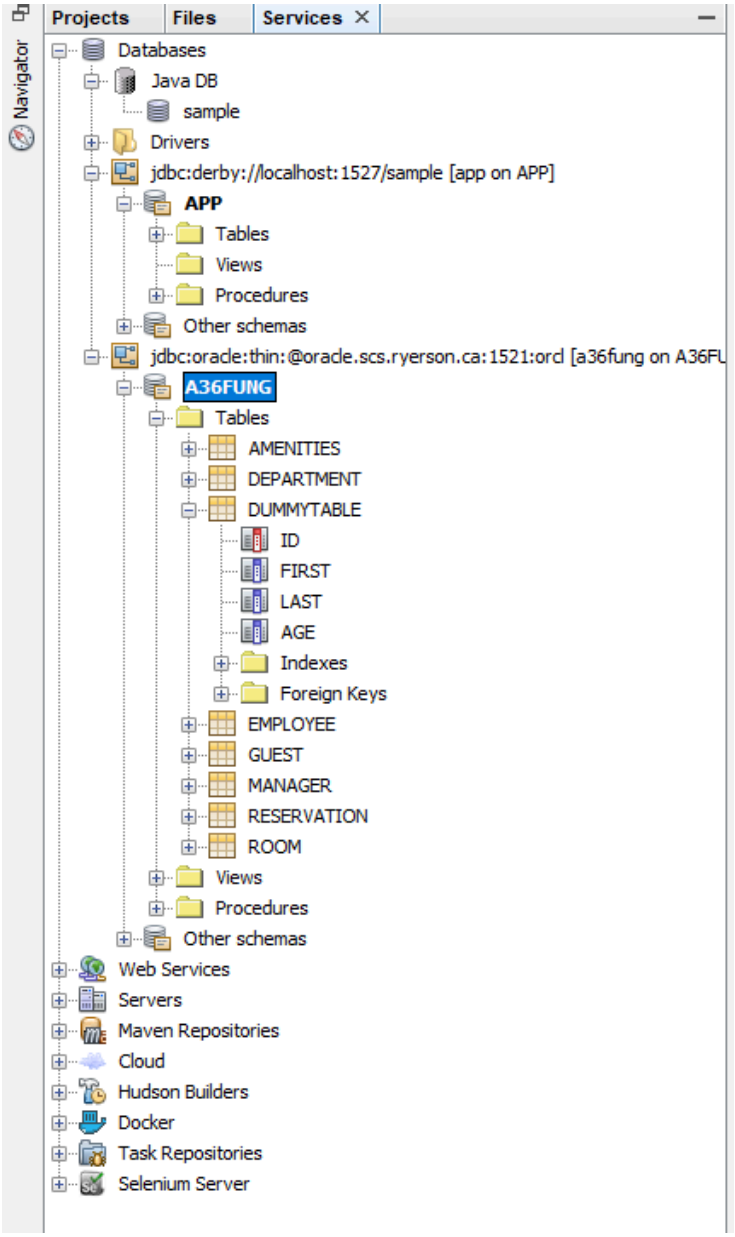


Figure 38: Screenshot of DummyTable added using Java code

Creating Tables:

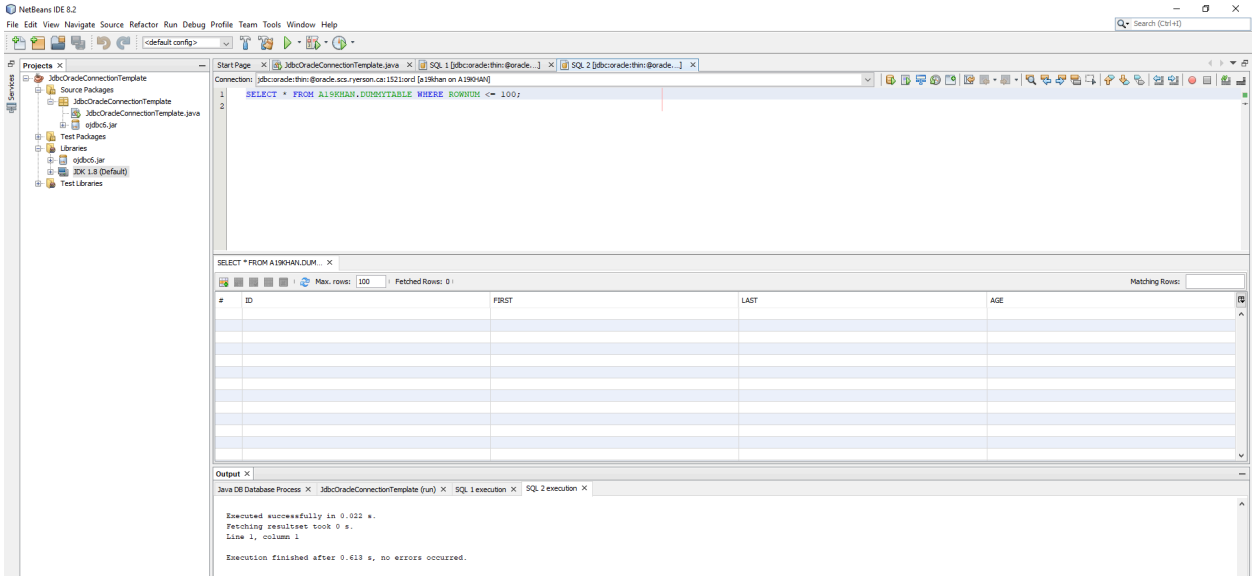


Figure 39: Creating dummy table using DUMMY table

Populating Tables:

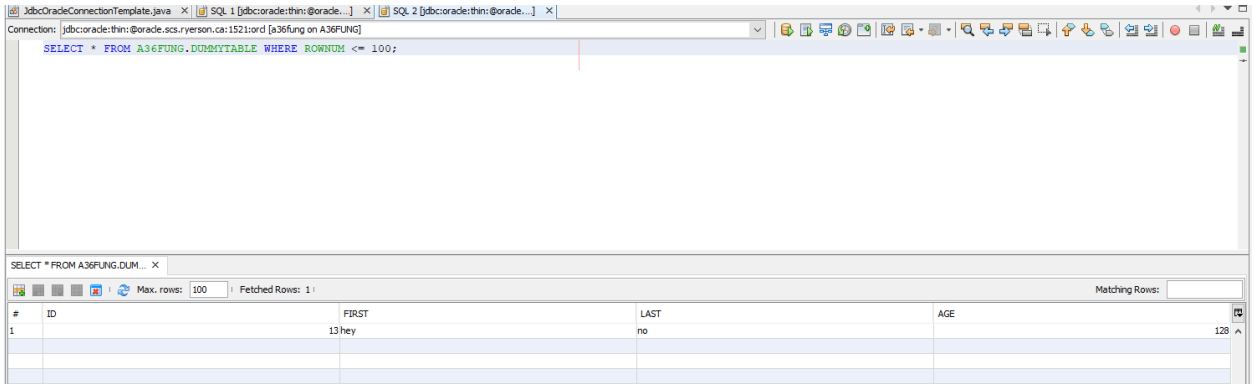


Figure 40: Demonstration of dummy data added to DUMMY table

Dropping Tables:

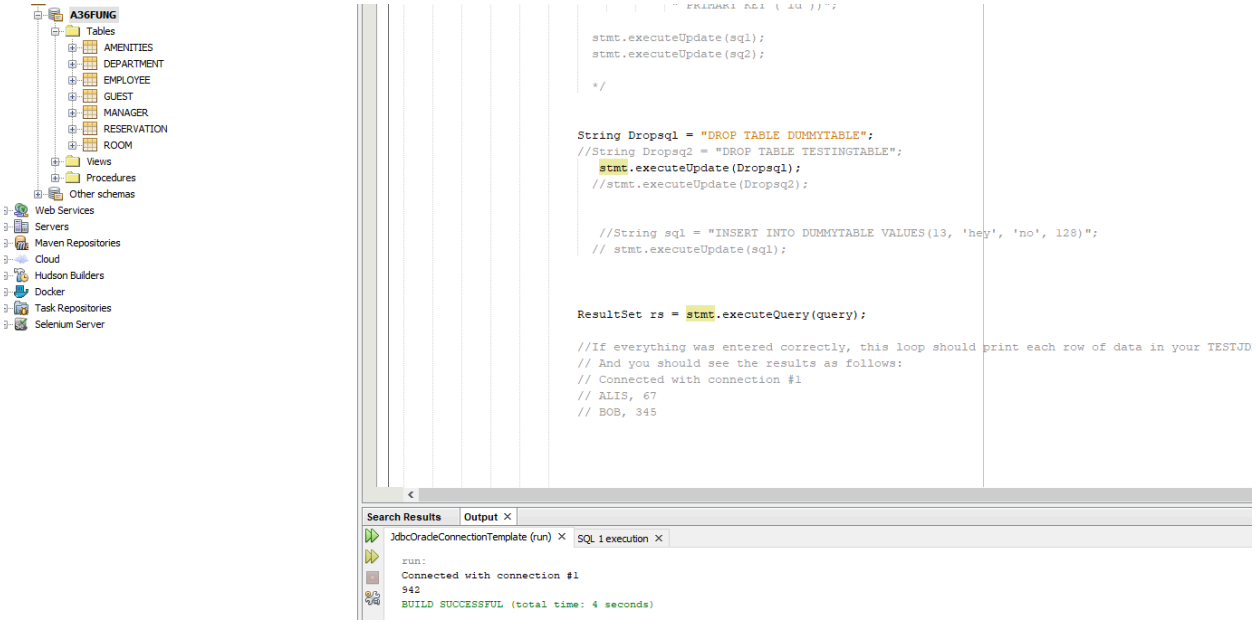


Figure 41: Dropping DUMMY table using java code