Edinburgh Napier UNIVERSITY

ADVANCED DATABASE SYSTEM CW1

WEIGUANG RAN
40412989@live.napier.ac.uk

SET09107

15 March 2019

1. Abstract

A bank has several branches in the UK. It needs a database to store information about its local branches. Each branch is identified by a unique branch code, an address (street, city, post code), and a phone number. The customer accounts at each branch are also recorded. Each customer account is identified by a unique account number, an account type (current or savings), and a balance.

Each account has an interest rate (interest rate can be determined by yourself - any reasonable one will be fine). An account is also associated with exactly one branch. The date when the account is opened is recorded as well.

An account must be classified as either a current or a savings account (but not both). A current account also has a limit of free overdraft (overdraft can be determined by yourself - any reasonable one will be fine). The free overdraft limit is set at the opening of an account. Data about customers and employees is also recorded. All customers and employees have an associated National Insurance number (a tax payer's unique identification number), address (street, city, post code) and phone numbers (home number and mobile numbers). An employee cannot be a customer at the same branch where he/she works.

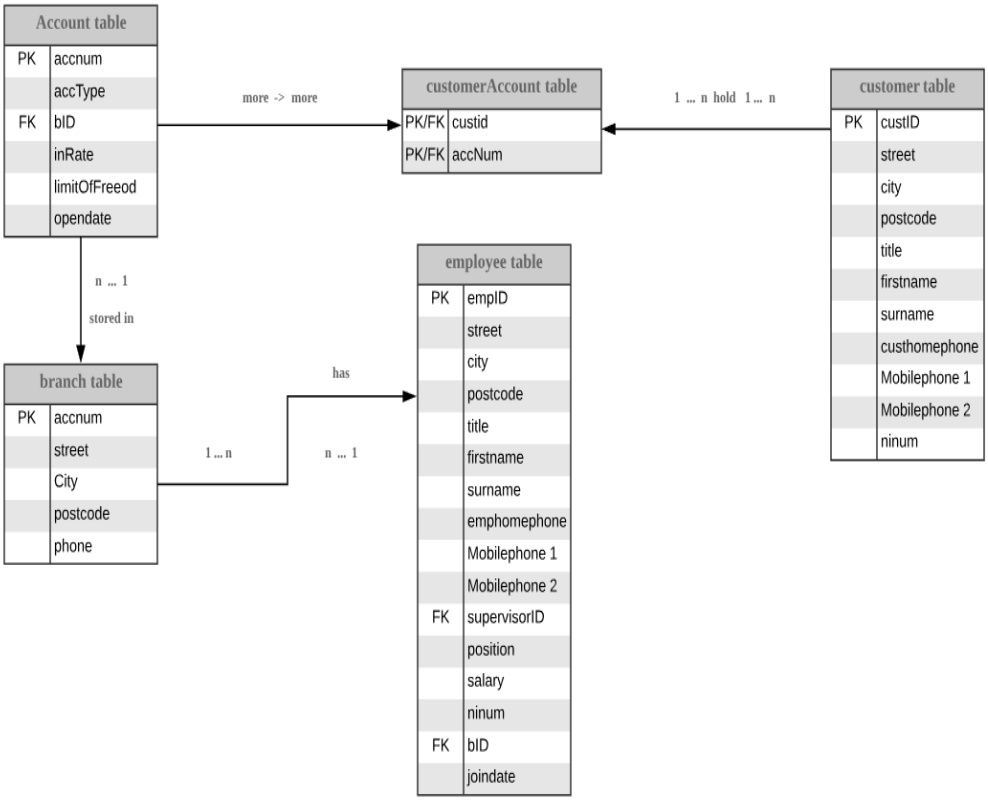
An employee has a job position (Head, Manager, Project Leader, Accountant, Cashier) and a salary, and works for exactly one branch. The date that the employee joined the bank is also recorded. Every employee has a supervisor at the same branch, except the head of the branch. The head of the branch is the only person who is not supervised by anyone at the same branch.

A customer may have multiple accounts with the bank, and an account may be owned by multiple customers as a joint account.

Key words: WEIGUANG RAN, SET09107, ORACLE 11g, sql developer, Database

2. Contents

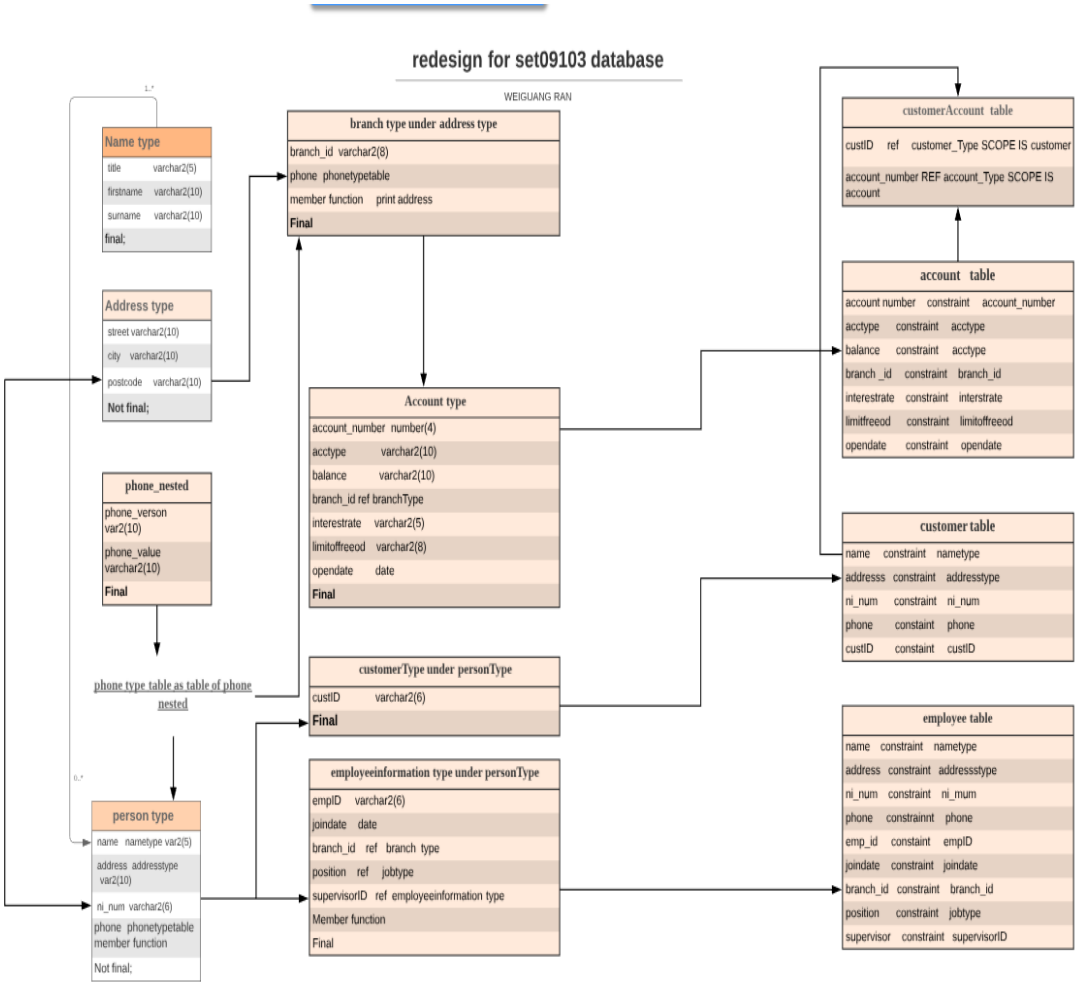
3. Entity Relational Diagram of my coursework



4. Re-design the database [The coding can be shown at task2 re-design.sql file]

Aim of re-design: The purpose of redesigning the database is to make the data import faster and more reasonable by creating reasonable objects and tables. In this process, the length, value and frequency of the data need to be considered. A good design is the key to the success of the database.

4.1Re-design flow picture



5.How to re-design the database

5.2.1 Data flow

Sub-type ---> Type ---> Sub-table ---> table

5.2.2 Type

Definition of object type: An **object type** is "*a user-defined composite datatype that encapsulates a data structure along with the functions and procedures needed to manipulate the data"(1).* Object type included two parts —— Object Type Specification and Object Type Body:

Object Type Specification : An interface between an object and an application that defines the public properties and methods of the object.

Object Type Body: used to implement the public method defined by the object type specification.

Why use object types: Object types can let people break down a large system into logical entities. *This let people create software components that are modular, maintainable, and reusable across projects and teams.(2)*It can help people define different object types and make each operate slightly differently. One more thing is that Object types allow for realistic data modeling.

Type in this database: In this database, originally I create 8 types and 5 tables with different properties, but when I insert data and doing test, I found jobs for different people made me confused, I need update the type and table, so I **add job type(have jobID, jobtitle and salary) and job table below job type**, then insert different jobs(head, manager, accounting and cashier into job table, this can be shown in my sql.doc. So finally I have 9 types and 6 tables.

5.2.3 References

Definition of ref: The purpose of ref takes as its argument a correlation variable(table alias) associated with a row of an object table or an object view. A ref value is returned for the object instance that is bound to the variables or row.(3) Reference is very strongly to use in object type, ref is kind of like foreign keys, taking information from other place(like type or other tables) to a new table, for example, in my database, when I want to use branchID data(in the branch type) in my account table, I can just write **select ref(b) from branch b where b.branch_id = 'xxx'.** And then I get get different data connect which the branchid which i write down.

5.2.4 Nested table

A nested table is a table in a table. A nested table is a collection of rows that are represented in the main table as one of them. A nested table can contain multiple rows for each record in the primary table. In a sense, it is a way to store a one-to-many relationship in a table. And in this database, I create phone numbers as the nested table, firstly, I create phone_nested type:

```
CREATE TYPE phone_nested AS OBJECT (
    phone_version VARCHAR2(20),
    phone_value VARCHAR2(20))
    FINAL;
```

Then,

```
CREATE TYPE phone_type_table AS TABLE OF phone_nested;
```

When I create table and want to store phone numbers, I can write coding just like:

```
CONSTRAINT phonenumber CHECK (phone IS NOT NULL),
NESTED TABLE phone STORE AS emp_phone_nested_table;
```

The characteristics of the nested table:

1. Object reuse: If you write object-oriented code, you have an increased chance of reusing previously written code modules. Similarly, if you create object-oriented database objects, you increase the chance that database objects can be reused.
2. Standard support: If you create standard objects, then their chances of being reused will increase.
3. Define the access path: For each object, the user can define the procedures and functions that run on it so that the data can be combined with the method of accessing the data. With access paths defined in this way, you can standardize data access methods and improve object reusability.

5.2.5 Constraint

*The purpose of constraint is that d*efining an ***integrity constraint****--a rule that restricts the values in a database. Normally there are six different types of constraint:(4)*

NOT NULL, Unique, primary key, foreign key, check and **REF**

People can define constraints syntactically in two ways: (1) As part of the definition of an individual column or attribute. This is called **inline** specification. (2) As part of the table definition. This is called **out-of-line** specification.

5.2.6 Object Methods

Object methods, also known as subprograms, are functions or procedures that you can declare in an object type definition to implement behavior that you want objects of that type to perform. An application calls the subprograms to invoke the behavior.

Subprograms can be written in PL/SQL or virtually any other programming language. Methods written in PL/SQL or Java are stored in the database. Methods written in other languages, such as C, are stored externally.(5)

When you create an object methods, there is a fixed format, like **Create Function xxx, Begin xxxx end;** for example, at my database, there is a type like this:

```
ALTER TYPE person_type
ADD MEMBER FUNCTION print_name RETURN STRING,
ADD MEMBER FUNCTION print_address RETURN STRING CASCADE;
/

CREATE OR REPLACE TYPE BODY person_type AS
MEMBER FUNCTION print_name RETURN STRING IS
    BEGIN
        RETURN name.title || '. ' || name.firstname || ' ' || name.surname;
    END print_name;

MEMBER FUNCTION print_address RETURN STRING IS
    BEGIN
        RETURN address.street || ', ' || address.city || ', ' || address.postcode;
    END print_address;
END;
/
```

Member function: member function is a special feature of oracle PL/SQL, this function is handle and run calculations on the data of the type that the ***member function*** is featured inside. And I use this method in my database for these things below:

(1) In the person type, people can select the full name of employees, not the first, title or second name, the member function can put them together and show up at the same time. **Coding can be found below the person_type.**

(2) In the address type, I create a full address function, just the same like full name, this function allow system to show the full address, like: **64 parkhead avenue, Edinburgh, EH114SE**, not like just **64 parkheadavenue**, or just **Edinburgh** or just **postcode. Coding can be found below the address_type.**

5.2.7 Varrays (Alternative design)

The varray (variable size array) is one of the three types of collections in PL/SQL (associative array, nested table, varray). The varray's key distinguishing feature is that when you declare a varray type, you specify the maximum number of elements that can be defined in the varray.

The different between varrays and nest table:

(1) Nested tables are unbounded, while mutable arrays have the largest size.

(2) Individual elements can be removed from a nested table, but they cannot be removed from the varray.

(3) Variables are stored online by Oracle (in the same tablespace), while nested table data is stored in a storage table, which is a system-generated database table associated with a nested table.

(4) Nested tables support indexes, while mutable arrays do not support indexes.

But finally in my database, I decide to use nest table compare with the varrays. The most important reason is that the nested table support indexes, because if the database used for the website, phone number must one thing which customer often search, so using nest table will improve a lot speed. And one more reason is that I am more familiar with nest table.

6. How the new database works

Most part of the types and tables are one to one relational, for example, just like employee_type under the employee table, the data which store in this type also only search by employee table. Customer type/Customer table, job type/job table the same as well. But the database also has one to more and more to more relational as well, for instance, phone type and person type is not one to one relational, the data which stored in there not only ref for one table, can be used for other tables as well. The table which named employee table job table and customer table are important, if an employee is a manager, he can be the supervisor to manage more people than a cashier or other lower level position works, if insert other position into the database. What is more, when I was doing the task, I found the method for assigning the awards need to refactored the table, so at last I add two new columns, which are used for the name of target tasks and the number of target tasks, also here I need to use if—else loop structure.

At first, I thought the job type is not necessary, I can just insert different job data, but when I insert data, I found the oracle cannot judge and allow insert more than one rows into one column at the same time, so I create the job type and job table to store the job data individually.

And the member function assigning awards to employees focus on their job functions. Which means different positon have different permission, the manager can be the supervisor for more than more staffs (except other managers). Also at task F, I created two new columns aimed to realize function, one column is for the target task, how many staffs for each manager. The other one is how many years the employee work in their branch, I use trunc to analysis the year.

Table screenshots [The coding for this task can be shown at Task3. insert.sql file]

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Sort.. | Filter:

	ACCOUNT_NUMBER	ACC_TYPE	BALANCE	BRANCH_ID	INTEREST_RATE	LIMIT_OF_FREE_OD	OPEN_DATE
1	1001	current	821	[SYSTEM.BRANCH_TYPE]	0.005	800	01-MAY-15
2	1010	saving	3122	[SYSTEM.BRANCH_TYPE]	.02	(null)	01-MAY-15
3	8002	current	821	[SYSTEM.BRANCH_TYPE]	0.005	100	01-MAY-10
4	1234	current	1000	[SYSTEM.BRANCH_TYPE]	0.023	600	14-JUN-97
5	1101	saving	9877	[SYSTEM.BRANCH_TYPE]	0.023	600	14-JUN-97
6	1102	saving	1235	[SYSTEM.BRANCH_TYPE]	0.01	10000	21-FEB-00
7	1103	saving	123	[SYSTEM.BRANCH_TYPE]	0.003	9999	16-FEB-00
8	1104	saving	145	[SYSTEM.BRANCH_TYPE]	0.013	444	02-MAR-00
9	1105	current	1444	[SYSTEM.BRANCH_TYPE]	0.05	100	04-APR-00
10	1106	saving	800	[SYSTEM.BRANCH_TYPE]	0.003	2208	28-FEB-10
11	1107	current	123	[SYSTEM.BRANCH_TYPE]	0.04	450	21-DEC-09
12	1108	saving	400	[SYSTEM.BRANCH_TYPE]	0.05	600	21-NOV-16
13	1109	current	7800	[SYSTEM.BRANCH_TYPE]	0.03	(null)	01-JAN-00
14	1110	saving	1235	[SYSTEM.BRANCH_TYPE]	0.01	10000	21-FEB-00
15	1111	saving	10000	[SYSTEM.BRANCH_TYPE]	0.09	500	16-JUN-09
16	1112	current	1220	[SYSTEM.BRANCH_TYPE]	0.002	200	03-MAR-18
17	1113	saving	88000	[SYSTEM.BRANCH_TYPE]	0.04	1000	15-JUL-19
18	1114	current	432	[SYSTEM.BRANCH_TYPE]	0.06	100	30-MAY-14
19	1115	current	3420000	[SYSTEM.BRANCH_TYPE]	0.01	9000	15-AUG-99
20	1116	saving	9888	[SYSTEM.BRANCH_TYPE]	0.08	700	30-JUN-03
21	1117	current	10000	[SYSTEM.BRANCH_TYPE]	0.012	5000	14-FEB-10
22	1118	saving	5600	[SYSTEM.BRANCH_TYPE]	0.014	5000	13-MAY-99
23	1119	current	9888	[SYSTEM.BRANCH_TYPE]	0.009	8000	02-AUG-98
24	1120	saving	2000	[SYSTEM.BRANCH_TYPE]	0.008	4000	17-JUN-07

Figure1: Account table

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Sort.. | Filter:

	STREET	CITY	POSTCODE	BRANCH_ID	PHONE
1	colinton st.	Edinbutgh	EH11 5DT	901	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED])
2	Market st.	Edinburgh	EH1 5AB	902	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED])
3	Bridge st.	Glasglow	G18 1QQ	903	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED])
4	Chao Yang st.	London	E11 W1	904	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED])
5	Queen st.	Edinburgh	EH6 5EE	905	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED])
6	RanTun st..	Edinburgh	EH4 5NE	906	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED])
7	Marry st.	Edinburgh	EH1 4QH	907	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED])
8	Ford st.	Edinburgh	EH8 2SE	908	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED])
9	Horry st.	Edinburgh	EH12 2ER	909	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED])
10	Royal Mill	Edinburgh	EH1 3AA	910	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED])
11	King st.	Edinburgh	EH3 5AR	911	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED])
12	Qinling St.	Zhengzhou	450001	912	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED])
13	Futian st.	Glasglow	450002	913	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED])
14	Wenhua st.	Beijing	400000	914	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED])
15	Tianan st.	Edinburgh	EH1 1RR	915	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED])
16	Yahoo st.	Oxford	11ER 3BT	917	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED])
17	Tianmao st.	Cambridge	CA1 AA3	918	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED])
18	Taobao	Striling	ST1 ER3	919	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED])
19	MeiTuan st.	Dumdee	3AD 4SE	920	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED])
20	Jingdong st.	Stirling	E12 R12	921	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED])

Figure2: branch table

Welcome Page | system | CUSTOMER

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Sort.. | Filter:

	NAME	ADDRESS	NI_NUM	PHONE	CUST_ID
1	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1003	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED])	1002
2	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1010	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED])	1003
3	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1034	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED])	1098
4	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1201	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED])	1088
5	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1120	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED])	1004
6	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1002	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED])	1005
7	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1004	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED])	1006
8	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1005	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED])	1007
9	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1007	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED])	1008
10	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1008	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED])	1009
11	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1012	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED])	1010
12	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1013	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED])	1011
13	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1088	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED])	1012
14	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1078	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED])	1013
15	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1079	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED])	1014
16	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1080	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED])	1015
17	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1081	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED])	1016
18	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1082	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED])	1017
19	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1083	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED])	1018
20	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1084	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED])	1019
21	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1085	SYSTEM.PHONE_TYPE_TABLE([SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED], [SYSTEM.PHONE_NESTED])	1020

Figure3: Customer table

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Sort.. Filter:

	NAME	ADDRESS	NI_NUM	PHONE	...	JOIN_DATE	BRANCH_ID	POSITION	SUPERVISOR_ID
1	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1001	SYSTEM.PHONE_TYPE_TABLE(...	101	01-FEB-06	[SYSTEM.BRANCH_TYPE]	[SYSTEM.JOB_TYPE]	(null)
2	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1010	SYSTEM.PHONE_TYPE_TABLE(...	105	04-MAR-07	[SYSTEM.BRANCH_TYPE]	[SYSTEM.JOB_TYPE]	[SYSTEM.EMPLOYEE_INFO_TYPE]
3	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1120	SYSTEM.PHONE_TYPE_TABLE(...	108	01-FEB-10	[SYSTEM.BRANCH_TYPE]	[SYSTEM.JOB_TYPE]	[SYSTEM.EMPLOYEE_INFO_TYPE]
4	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1002	SYSTEM.PHONE_TYPE_TABLE(...	666	11-NOV-11	[SYSTEM.BRANCH_TYPE]	[SYSTEM.JOB_TYPE]	[SYSTEM.EMPLOYEE_INFO_TYPE]
5	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1004	SYSTEM.PHONE_TYPE_TABLE(...	801	04-MAR-07	[SYSTEM.BRANCH_TYPE]	[SYSTEM.JOB_TYPE]	(null)
6	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1003	SYSTEM.PHONE_TYPE_TABLE(...	802	05-FEB-14	[SYSTEM.BRANCH_TYPE]	[SYSTEM.JOB_TYPE]	[SYSTEM.EMPLOYEE_INFO_TYPE]
7	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1005	SYSTEM.PHONE_TYPE_TABLE(...	206	07-JUN-14	[SYSTEM.BRANCH_TYPE]	[SYSTEM.JOB_TYPE]	(null)
8	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1007	SYSTEM.PHONE_TYPE_TABLE(...	203	04-MAR-07	[SYSTEM.BRANCH_TYPE]	[SYSTEM.JOB_TYPE]	(null)
9	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1008	SYSTEM.PHONE_TYPE_TABLE(...	202	13-SEP-12	[SYSTEM.BRANCH_TYPE]	[SYSTEM.JOB_TYPE]	[SYSTEM.EMPLOYEE_INFO_TYPE]
10	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1009	SYSTEM.PHONE_TYPE_TABLE(...	302	04-MAR-07	[SYSTEM.BRANCH_TYPE]	[SYSTEM.JOB_TYPE]	[SYSTEM.EMPLOYEE_INFO_TYPE]
11	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1011	SYSTEM.PHONE_TYPE_TABLE(...	402	15-JAN-14	[SYSTEM.BRANCH_TYPE]	[SYSTEM.JOB_TYPE]	[SYSTEM.EMPLOYEE_INFO_TYPE]
12	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1012	SYSTEM.PHONE_TYPE_TABLE(...	502	15-MAR-17	[SYSTEM.BRANCH_TYPE]	[SYSTEM.JOB_TYPE]	[SYSTEM.EMPLOYEE_INFO_TYPE]
13	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1013	SYSTEM.PHONE_TYPE_TABLE(...	602	15-DEC-07	[SYSTEM.BRANCH_TYPE]	[SYSTEM.JOB_TYPE]	[SYSTEM.EMPLOYEE_INFO_TYPE]
14	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1014	SYSTEM.PHONE_TYPE_TABLE(...	107	14-MAY-15	[SYSTEM.BRANCH_TYPE]	[SYSTEM.JOB_TYPE]	[SYSTEM.EMPLOYEE_INFO_TYPE]
15	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1015	SYSTEM.PHONE_TYPE_TABLE(...	204	04-MAR-07	[SYSTEM.BRANCH_TYPE]	[SYSTEM.JOB_TYPE]	[SYSTEM.EMPLOYEE_INFO_TYPE]
16	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1016	SYSTEM.PHONE_TYPE_TABLE(...	303	05-JUL-16	[SYSTEM.BRANCH_TYPE]	[SYSTEM.JOB_TYPE]	[SYSTEM.EMPLOYEE_INFO_TYPE]
17	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1017	SYSTEM.PHONE_TYPE_TABLE(...	304	15-AUG-16	[SYSTEM.BRANCH_TYPE]	[SYSTEM.JOB_TYPE]	[SYSTEM.EMPLOYEE_INFO_TYPE]
18	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1018	SYSTEM.PHONE_TYPE_TABLE(...	403	12-JUN-13	[SYSTEM.BRANCH_TYPE]	[SYSTEM.JOB_TYPE]	[SYSTEM.EMPLOYEE_INFO_TYPE]
19	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1019	SYSTEM.PHONE_TYPE_TABLE(...	404	16-JUN-15	[SYSTEM.BRANCH_TYPE]	[SYSTEM.JOB_TYPE]	[SYSTEM.EMPLOYEE_INFO_TYPE]
20	[SYSTEM.NAME_TYPE]	[SYSTEM.ADDRESS_TYPE]	N1020	SYSTEM.PHONE_TYPE_TABLE(...	405	16-AUG-15	[SYSTEM.BRANCH_TYPE]	[SYSTEM.JOB_TYPE]	[SYSTEM.EMPLOYEE_INFO_TYPE]

Figure4: Employee table

	CUST_ID	ACCOUNT_NUMBER
1	[SYSTEM.CUSTOMER_TYPE]	(null)
2	[SYSTEM.CUSTOMER_TYPE]	[SYSTEM.ACCOUNT_TYPE]
3	[SYSTEM.CUSTOMER_TYPE]	[SYSTEM.ACCOUNT_TYPE]
4	[SYSTEM.CUSTOMER_TYPE]	[SYSTEM.ACCOUNT_TYPE]
5	[SYSTEM.CUSTOMER_TYPE]	[SYSTEM.ACCOUNT_TYPE]
6	[SYSTEM.CUSTOMER_TYPE]	[SYSTEM.ACCOUNT_TYPE]
7	[SYSTEM.CUSTOMER_TYPE]	[SYSTEM.ACCOUNT_TYPE]
8	[SYSTEM.CUSTOMER_TYPE]	[SYSTEM.ACCOUNT_TYPE]
9	[SYSTEM.CUSTOMER_TYPE]	[SYSTEM.ACCOUNT_TYPE]
10	[SYSTEM.CUSTOMER_TYPE]	[SYSTEM.ACCOUNT_TYPE]
11	[SYSTEM.CUSTOMER_TYPE]	[SYSTEM.ACCOUNT_TYPE]
12	[SYSTEM.CUSTOMER_TYPE]	(null)
13	[SYSTEM.CUSTOMER_TYPE]	[SYSTEM.ACCOUNT_TYPE]
14	[SYSTEM.CUSTOMER_TYPE]	(null)
15	[SYSTEM.CUSTOMER_TYPE]	[SYSTEM.ACCOUNT_TYPE]
16	[SYSTEM.CUSTOMER_TYPE]	[SYSTEM.ACCOUNT_TYPE]
17	[SYSTEM.CUSTOMER_TYPE]	[SYSTEM.ACCOUNT_TYPE]
18	[SYSTEM.CUSTOMER_TYPE]	[SYSTEM.ACCOUNT_TYPE]
19	[SYSTEM.CUSTOMER_TYPE]	[SYSTEM.ACCOUNT_TYPE]
20	[SYSTEM.CUSTOMER_TYPE]	[SYSTEM.ACCOUNT_TYPE]
21	[SYSTEM.CUSTOMER_TYPE]	[SYSTEM.ACCOUNT_TYPE]
22	[SYSTEM.CUSTOMER_TYPE]	[SYSTEM.ACCOUNT_TYPE]

Figure5: Customer_account table

7.Questions & Answers

[This task coding can be shown at task4 questions .sql file]

7.1 Question1:

```
SELECT
e.name.title
|| '. ' ||
                e.name.firstName || ' ' ||
                e.name.surname || ' is living in ' ||
                e.address.city
            AS  "Employees: 'ar' and  Edinburgh"
            FROM    employee e
            WHERE   INSTR(e.name.firstname,'ar')>0
            AND e.address.city = 'Edinburgh';
```

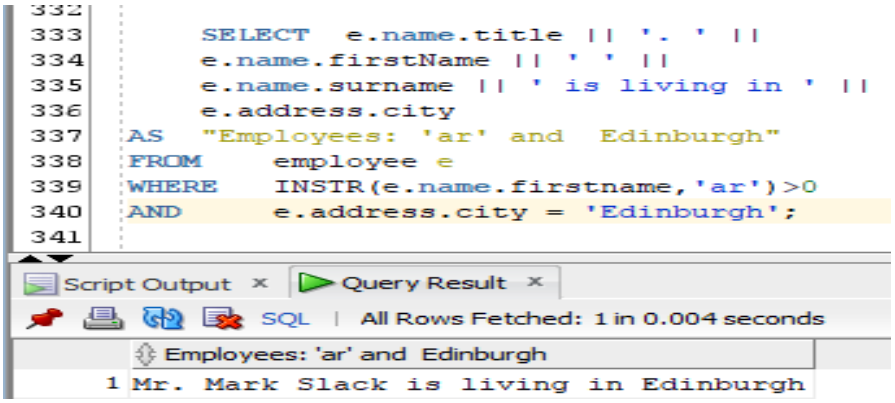


Figure6: Question1

7.2 Question2:

```
SELECT
                a.branch_id.branch_id AS "Branch ID",
                a.branch_id.city AS "Address",
                count(a.acc_type) AS "Number of savingsaccounts"
            FROM
                account a
            WHERE
                acc_type = 'saving'
            GROUP BY
                a.acc_type, a.branch_id.city, a.branch_id.branch_id
            ORDER BY
                a.branch_id.branch_id ASC;
```

```
320     SELECT
321       a.branch_id.branch_id AS "Branch ID",
322       a.branch_id.city AS "Address",
323       count(a.acc_type) AS "Number of savingsaccounts"
324   FROM
325       account a
326   WHERE
327       acc_type = 'saving'
328   GROUP BY
329       a.acc_type, a.branch_id.city, a.branch_id.branch_id
330   ORDER BY
331       a.branch_id.branch_id ASC;
332
```

Script Output | Query Result | SQL | All Rows Fetched: 9 in 0.004 seconds

	Branch ID	Address	Number of savingsaccounts
1	901	Edinbutgh	1
2	903	Glasglow	1
3	904	London	2
4	907	Edinburgh	2
5	909	Edinburgh	2
6	912	Zhengzhou	1
7	914	Beijing	2
8	915	Edinburgh	1
9	917	Oxford	1

Figure7: Question2

7.3 Question3:

```
ALTER TYPE person_type
ADD MEMBER FUNCTION print_name RETURN STRING,
ADD MEMBER FUNCTION print_address RETURN STRING CASCADE;
/

CREATE OR REPLACE TYPE BODY person_type AS
MEMBER FUNCTION print_name RETURN STRING IS
    BEGIN
        RETURN name.title || '. ' || name.firstname || ' ' || name.surname;
    END print_name;


SELECT
        c.account_number.branch_id.branch_id AS branchID,
        c.cust_id.print_name() AS names,
        c.account_number.balance AS balance
    FROM (
        SELECT
            c.account_number.branch_id.branch_id AS branch_id,
            c.account_number.acc_type AS acc_type,
            MAX (c.account_number.balance) AS max_balance
            FROM
```

```
            customer_account c
        WHERE
            c.account_number.acc_type = 'saving'
        GROUP BY c.account_number.branch_id.branch_id, c.account_number.acc_type
    ) balance
JOIN customer_account c
ON
    c.account_number.branch_id.branch_id = balance.branch_id
AND
    c.account_number.acc_type = balance.acc_type
AND
    c.account_number.balance = balance.max_balance
LEFT JOIN customer_account t2
ON t2.cust_id.cust_id = c.cust_id.cust_ID
AND t2.account_number.acc_type = 'Basic';
```

```
297     SELECT
298         c.account_number.branch_id.branch_id AS branchID,
299         c.cust_id.print_name() AS names,
300         c.account_number.balance AS balance
301     FROM (
302         SELECT
303             c.account_number.branch_id.branch_id AS branch_id,
304             c.account_number.acc_type AS acc_type,
305             MAX (c.account_number.balance) AS max_balance
306             FROM
307                 customer_account c
308             WHERE
309                 c.account_number.acc_type = 'saving'
310             GROUP BY c.account_number.branch_id.branch_id, c.account_number.acc_type
311         ) balance
312     JOIN customer_account c
313     ON c.account_number.branch_id.branch_id = balance.branch_id
314     AND  c.account_number.acc_type = balance.acc_type
315     AND  c.account_number.balance = balance.max_balance
316     LEFT JOIN customer_account t2
317     ON t2.cust_id.cust_id = c.cust_id.cust_ID
318     AND t2.account_number.acc_type = 'Basic';
```

Script Output | Query Result | SQL | All Rows Fetched: 8 in 0.014 seconds

	BRANCHID	NAMES	BALANCE
1	901	Mr. Jack Smith	3122
2	901	Mrs. Anna Smith	3122
3	901	Mrs. Anna Smith	3122
4	904	Mr. Peter Bill	9888
5	915	Mr. Yifu Lai	5600
6	912	Mr. Frank David	10000
7	907	Mr. Neil Fed	88000
8	904	Mrs. May Bery	9888

Figure8: Question3

7.4 Question4:

```
SELECT
    'Emp ID: ' || e.emp_id || ',    ' ||
    e.print_name()      AS employee,
    c.account_number.account_number AS "Account Number",
    e.supervisor_id.print_name() AS "Supervisor",
    e.supervisor_id.position.job_title AS "Supervisor Job Title"
```

```
FROM
    employee e, customer_account c
WHERE
    c.cust_id.name.firstname = e.name.firstname
AND
    c.cust_id.name.surname = e.name.surname
AND
    e.supervisor_id.position.job_title = 'Manager'
ORDER BY
    e.emp_id ASC;
```

```
281     'Emp ID: ' || e.emp_id || ',  ' ||
282     e.print_name()   AS employee,
283     c.account_number.account_number AS "Account Number",
284     e.supervisor_id.print_name() AS "Supervisor",
285     e.supervisor_id.position.job_title AS "Supervisor Job Title"
286 FROM
287     employee e, customer_account c
288 WHERE
289     c.cust_id.name.firstname = e.name.firstname
290 AND
291     c.cust_id.name.surname = e.name.surname
292 AND
293     e.supervisor_id.position.job_title = 'Manager'
294 ORDER BY
295     e.emp_id ASC;
```

Script Output | Query Result | SQL | All Rows Fetched: 3 in 0.005 seconds

	EMPLOYEE	Account Number	Supervisor	Supervisor Job Title
1	Emp ID: 108, Mr. Mark Slack	8002	Mr. John William	Manager
2	Emp ID: 802, Mr. Jack Smith	(null)	Mrs. July William	Manager
3	Emp ID: 802, Mr. Jack Smith	1010	Mrs. July William	Manager

Figure9: Question4

7.5 Question5:

```
SELECT
    c.account_number.branch_id.branch_id AS branch_id,
    c.cust_id.print_name() AS full_name,
    c.account_number.limit_of_free_OD AS free_od
    FROM (
        SELECT c.account_number.branch_id.branch_id AS branch_id,
            MAX(c.account_number.limit_of_free_OD) AS maxOD
            FROM customer_account c
            GROUP BY c.account_number.branch_id.branch_id
        ) maxOD, customer_account c
        WHERE c.account_number.limit_of_free_OD = maxOD.maxOD AND
          c.account_number.branch_id.branch_id = maxOD.branch_id
          ORDER BY c.account_number.branch_id.branch_id ASC;
```

```
SELECT
   c.account_number.branch_id.branch_id AS branch_id,
   c.cust_id.print_name() AS full_name,
   c.account_number.limit_of_free_OD AS free_od
   FROM (
       SELECT c.account_number.branch_id.branch_id AS branch_id,
           MAX(c.account_number.limit_of_free_OD) AS maxOD
           FROM customer_account c
           GROUP BY c.account_number.branch_id.branch_id
       ) maxOD, customer_account c
       WHERE c.account_number.limit_of_free_OD = maxOD.maxOD AND
        c.account_number.branch_id.branch_id = maxOD.branch_id
        ORDER BY c.account_number.branch_id.branch_id ASC;
```

Script Output | Query Result

SQL | All Rows Fetched: 15 in 0.008 seconds

	BRANCH_ID	FULL_NAME	FREE_OD
1	901	Mrs. April Marry	600
2	901	Mr. Niuniu Bee	600
3	904	Mrs. May Bery	700
4	904	Mr. Peter Bill	700
5	907	Mr. Neil Fed	1000
6	908	Mr. Mark Slack	100
7	909	Mrs. July William	9000
8	909	Mrs. Millie Canny	9000
9	911	Mr. WEIGUANG RAN	100
10	911	Mrs. June Lucy	100
11	911	Mrs. Timi Cram	100
12	912	Mr. Frank David	500

Figure10: Question5

7.6 Question6:

```
SELECT
    c.print_name() as fullname,
    t.phone_value as phonenumber
FROM
    (SELECT c.cust_id AS cust_id, count(t.phone_version) AS mob_count, phone_version AS
phone_version
    FROM customer c, table(c.phone) t
    WHERE t.phone_version = 'Mobile'
    AND t.phone_value LIKE '0750%'
    GROUP BY c.cust_id, phone_version) phone_nums, customer c, table(c.phone) t
WHERE c.cust_id = phone_nums.cust_id
AND t.phone_version = phone_nums.phone_version
AND t.phone_version = 'Mobile'
AND phone_nums.mob_count > 0
ORDER BY c.cust_id;
```

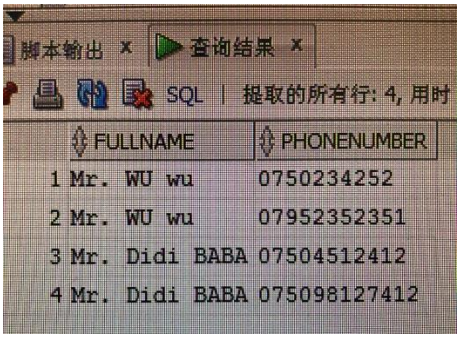


Figure11:Question6

7.7Question7:

```
SELECT
    COUNT(e.print_name()) as number_of_employees,
    (SELECT e.supervisor_id.print_name() FROM employee_table e WHERE e.supervisor_id.name.Surname
= 'Smith') AS supervisor_by_Mr_Smith
    FROM employee_table e
    WHERE
        e.supervisor_id.name.firstname = 'Jack' AND
        e.supervisor_id.name.surname = 'Smith'

    GROUP BY e.supervisor_id.print_name() ;

SELECT
    COUNT(e.print_name()) as number_of_employees,
    (SELECT e.supervisor_id.print_name() FROM employee_table e WHERE
e.supervisor_id.name.firstname = 'Jones') AS supervisor_by_Mrs_Jones
    FROM employee_table e
    WHERE
        e.supervisor_id.name.firstname = 'Jones' AND
        e.supervisor_id.name.surname = 'Wills'

    GROUP BY e.supervisor_id.print_name() ;
```

7.8 Question8:

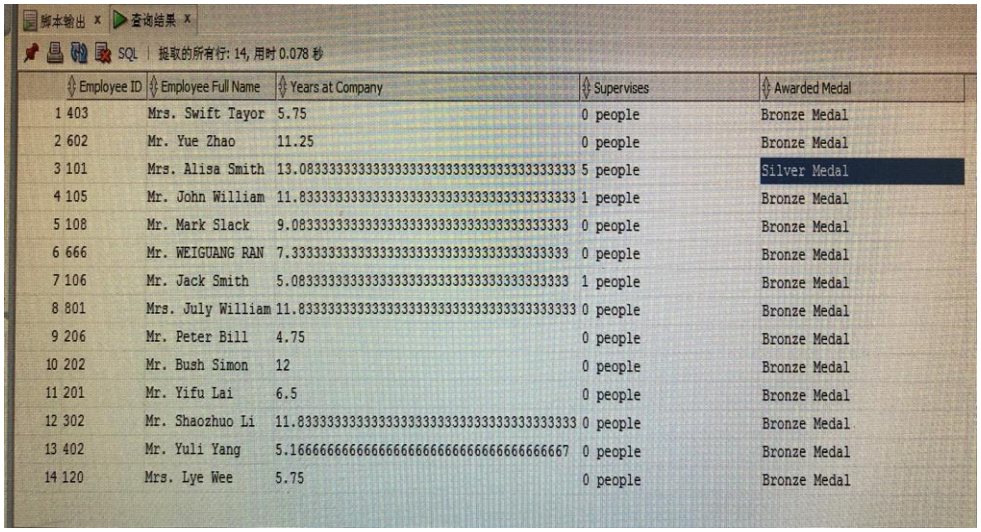
```
CREATE OR REPLACE TYPE BODY employee_info_Type AS
member function awardStar return varchar2 is
    medal varchar2(15);
    years number;
    emps number;
    begin
        select count(*)
        into     emps
        from     employee e
        where    Deref(e.supervisor_ID).emp_ID = self.emp_ID;
        years := trunc(months_between(to_date('19-03-19','dd-mm-yyyy'),self.join_Date))/12;
       if years > 12 AND emps > 6 then
            medal := 'Gold Medal';
            elsif years > 8 AND emps > 3 then
            medal := 'Silver Medal';
            elsif years > 4 then
            medal := 'Bronze Medal';
            else
            medal := 'No Medal Awarded';
        end if;
```

```
        return medal;
    end awardStar;

    member function years_at_company RETURN varchar2 IS
    years number;
    begin
        years := trunc(months_between(to_date('19-03-19','dd-mm-yyyy'),self.join_Date))/12;
        return years;
    end years_at_company;

    member function count_supervised RETURN varchar2 IS
    emps number;
    begin
        select count(*)
        into    emps
        from    employee e
        where   Deref(e.supervisor_ID).emp_ID = self.emp_ID;

        return emps;
    end count_supervised;
end;
/
SELECT
    e.emp_id AS "Employee ID" ,
    e.print_name() AS "Employee Full Name" ,
    e.years_at_company() AS "Years at Company",
    e.count_supervised() || ' people' AS "Supervises",
    e.awardStar() AS "Awarded Medal"
FROM
    employee e WHERE e.awardStar() != 'No Medal Awarded';
```



脚本输出 x 查询结果 x

SQL | 提取的所有行: 14, 用时 0.078 秒

	Employee ID	Employee Full Name	Years at Company	Supervises	Awarded Medal
1	403	Mrs. Swift Tayor	5.75	0 people	Bronze Medal
2	602	Mr. Yue Zhao	11.25	0 people	Bronze Medal
3	101	Mrs. Alisa Smith	13.083333333333333333333333333333333333	5 people	Silver Medal
4	105	Mr. John William	11.833333333333333333333333333333333333	1 people	Bronze Medal
5	108	Mr. Mark Slack	9.0833333333333333333333333333333333333	0 people	Bronze Medal
6	666	Mr. WEIGUANG RAN	7.3333333333333333333333333333333333333	0 people	Bronze Medal
7	106	Mr. Jack Smith	5.0833333333333333333333333333333333333	1 people	Bronze Medal
8	801	Mrs. July William	11.833333333333333333333333333333333333	0 people	Bronze Medal
9	206	Mr. Peter Bill	4.75	0 people	Bronze Medal
10	202	Mr. Bush Simon	12	0 people	Bronze Medal
11	201	Mr. Yifu Lai	6.5	0 people	Bronze Medal
12	302	Mr. Shaozhuo Li	11.833333333333333333333333333333333333	0 people	Bronze Medal
13	402	Mr. Yuli Yang	5.166666666666666666666666666666666667	0 people	Bronze Medal
14	120	Mrs. Lye Wee	5.75	0 people	Bronze Medal

Figure12: Question8

8. Critical discussion

Advantages of the object-relational database:

(1) Have extended data types – Object-relational database system allow users to define a new data type and corresponding operations based on application requirements. The new data types, once defined, is shared by all users just like the basic data type. For bank database, data always need update and have the new data type, so the object-relational database would be easier to change for DBA.

(2) Support for complex objects-the object-relational database system can support complex objects in SQL, and implement processing such as querying complex objects. A complex object is an object that consists of multiple basic types or user-defined data types. Considering the database of bank always have many members and complex data types, it would be the best choice.

(3) Support the concept of inheritance. Inheritance is an important concept of object-oriented technology. The object-relational database system can support the concept of subtype and super type, that means, support the concept of inheritance, such as the inheritance of attribute data and the inheritance of functions and procedures; With multiple inheritance, etc. It also supports important object-oriented ideas such as function overloading.

(4) Provide a general rule system. Object-relational database system provides a powerful and versatile rules system. In traditional, it is generally triggered to ensure the integrity of the data in the database. The trigger is a form of the rule. This advantage can help different DBA to maintain the database.

Advantage of the relational database:

(1) Maintain data consistency, data relational model based on relational model, structured storage, integrity constraint.

(2) Due to the premise of the premise of the standard, the cost of data update is small(the same filed is basically only one place);

(3) It is possible to perform complex queries such as join; This advantage is very Important for big database(like bank), because they usually need some complex queries including paging query and multi query.

(4) There are many practical results and professional technical information (mature technology). Many database using sql relational language, like Mysql, db2, sql2000.

Disadvantages of the objected-relational database:

Although objected-relational database model have many advantages, it looks like have both advantages of objected database and relational database. However, it just 'look like'. The advantages also can be the disadvantages, because it combined with two different databases and take advantages of their capability, so sometimes it will become very complex and difficult to handle. As for the database of bank, it means when DBA changed or left, the new DBA would need a long time to familiar with the former database.

Disadvantages of the relational database:

(1) Transaction consistency is not necessary nowadays: Relational databases have a large overhead in maintaining the consistency of things. Nowadays, many web2.0 systems do not have high read/write consistency on transactions, and transactions consistency is not so important now.

(2) Fixed table structure: very poor scalability, system upgrades, and increased functionality often mean huge changes in the data structure. This relational database is also difficult to cope with and requires

new structured data storage.

(3) Complex SQL: any large data volume web system is very jealous of multiple large table association queries, as well as complex data analysis types of complex SQL report queries. And often the thing is that the primary key query of a single table, as well as a simple conditional paged query of a single table, the function of SQL is greatly weakened.

Different between object-relational database and relational database:

(1) Method: Method is a special function which used for object-relational db but the relational db does not have it. A method is defined as "procedure or function that can operate on the attribute type."(5) There are different kinds of functions in method, in this database, I use member function to show the full name and full address.

(2) Types: Only object-relational database have types, often types are below the table, one type can contain many attributes, and the relational database just contain attributes in tables. Types have benefits and disadvantages, but just for bank database, using types is better.

9. Drop everything [Coding can be shown at task5.sql]

```
****
DROP EVERYTHING
****

*/

DROP TYPE name_Type FORCE;
DROP TYPE address_Type FORCE;
DROP TYPE branch_type FORCE;
DROP TYPE person_type FORCE;
DROP TYPE job_Type FORCE;
DROP TYPE employee_info_type FORCE;
DROP TYPE customer_type FORCE;
DROP TYPE account_type FORCE;
DROP TYPE phone_nested FORCE;
DROP TYPE phone_type_table FORCE;
DROP TABLE job_table;
DROP TABLE employee_table;
DROP TABLE account_table;
DROP TABLE branch_table;
DROP TABLE customer_table;
DROP TABLE customer_account;
```

10. Conclusion

This coursework is used for build a bank database in oracle 11g, and the main form used in this coursework is object-relational database. After this coursework, I understand the forms, methods, demand and the flow in oracle 11g. Also I considered and compared the advantages and disadvantages with relational database and object-relational database.

Complex 1

Reference:

1) **PL/SQL User's Guide and Reference**

   10*g* Release 1 (10.1)

   ,2003,(https://docs.oracle.com/cd/B14117_01/appdev.101/b10807/13_elems031.htm**)**

2) **Oracle® Database Performance Tuning Guide**
   11*g* Release 2 (11.2)---Performance Improvement Methods, 2010,
   (https://docs.oracle.com/cd/E18283_01/server.112/e16638/technique.htm)

3) **Database SQL language Reference, 2019,**
   (https://docs.oracle.com/cd/B28359_01/server.111/b28286/toc.htm)

4) **Oracle® Database PL/SQL Language Reference**
   11*g* Release 2 (11.2), 2010,
   (https://docs.oracle.com/cd/E18283_01/appdev.112/e17126/datatypes.htm#CJAEDAEA)

5) **Oracle Database 11g the complete reference, Kevin Lonely, 15 October 2008, Published by McGraw-Hill Education**

6) **Oracle FAQ's Nested table, 2018,**
   (http://www.orafaq.com/wiki/NESTED_TABLE**)**

7) **Database Advanced Application Developer's Guide---using PL/Scope**
   (https://docs.oracle.com/cd/B28359_01/appdev.111/b28424/adfns_plscope.htm#g1010526**)**

Complex 2 Files

In my zip. Files, there are these docs.:

1) SQL docs: Redesign.sql Insert.sql Qustions.sql Drop everything.sql
2) Report pdf --- the report for this coursework
3) Screenshots ---the pictures for each questions and tables