Part I – Working with an existing database

• Setting up Oracle Chinook

In this section you will begin the process of working with the Oracle Chinook database

Task – Open the Chinook\_Oracle.sql file and execute the scripts within.

2.0 SQL Queries

In this section you will be performing various queries against the Oracle Chinook database.

2.1 SELECT

Task – Select all records from the Employee table.

SELECT \* FROM employee;

Task – Select all records from the Employee table where last name is King.

SELECT \* FROM employee WHERE lastname = 'King';

Task – Select all records from the Employee table where first name is Andrew and REPORTSTO is NULL.

SELECT \* FROM employee WHERE firstname = 'Andrew' AND reportsto = NULL;

2.2 ORDER BY

Task – Select all albums in Album table and sort result set in descending order by title.

SELECT \* FROM album ORDER BY title DESC;

Task – Select first name from Customer and sort result set in ascending order by city

SELECT customer.firstname FROM customer ORDER BY city;

2.3 INSERT INTO

Task – Insert two new records into Genre table

INSERT INTO Genre (genreid, name) VALUES (26, Bluegrass);

INSERT INTO Genre (genreid, name) VALUES(27, Honky-Tonk);

Task – Insert two new records into Employee table

INSERT INTO employee (employeeid, lastname, firstname, title, reportsto, birthdate, hiredate, address, city, state, country, postalcode, phone, fax, email)

VALUES (9, 'Lancaster', 'Lewis', 'General Manager', 8, CURRENT\_TIMESTAMP, CURRENT\_TIMESTAMP, '1529 greatwell st', 'Memphis', 'TN', 'United States', 'T55 6G2', '+1(901)259-7786', '+1(901)767-8922', 'imnotreal@mail.com');

INSERT INTO employee (employeeid, lastname, firstname, title, reportsto, birthdate, hiredate, address, city, state, country, postalcode, phone, fax, email)

VALUES (10, 'Johnson', 'Marcus', 'General Manager', 5, CURRENT\_TIMESTAMP, CURRENT\_TIMESTAMP, '3209 filty st', 'Memphis', 'TN', 'United States', 'T46 789', '+1(901)778-7786', '+1(901)223-8542', 'imnotreal901@mail.com');

Task – Insert two new records into Customer table

INSERT INTO customer(customerid, firstname, lastname, company, address, city, state, country, postalcode, phone, fax, email, supportrepid)

VALUES (66, 'malcom', 'martin', '', '854 Pope St', 'Memphis', 'TN', 'United States', '38112', '+1(901)359-8576', '', 'idontknow@mail.com', 8 ); 2.4 UPDATE

INSERT INTO customer(customerid, firstname, lastname, company, address, city, state, country, postalcode, phone, fax, email, supportrepid)

VALUES (75, 'nwikpo', 'precious', '', '5979 Eggleston St', 'Memphis', 'TN', 'United States', '38126', '+1(901)444-0006', '', 'idontyou@mail.com', 7 );

Task – Update Aaron Mitchell in Customer table to Robert Walter

UPDATE customer SET lastname = 'Walter', firstname = 'Robert' WHERE customerid = 32;

Task – Update name of artist in the Artist table “Creedence Clearwater Revival” to “CCR”

UPDATE artist SET name = 'CCR' WHERE artistid = 76;

2.5 LIKE

Task – Select all invoices with a billing address like “T%”

SELECT \* FROM invoice WHERE billingaddress LIKE '%T%';

2.6 BETWEEN

Task – Select all invoices that have a total between 15 and 50

SELECT \* FROM invoice WHERE total BETWEEN 15 and 50;

Task – Select all employees hired between 1st of June 2003 and 1st of March 2004

SELECT \* FROM employee WHERE hiredate BETWEEN '2003-06-01' and '2004-03-01';

2.7 DELETE

Task – Delete a record in Customer table where the name is Robert Walter (There may be constraints that rely on this, find out how to resolve them).

DELETE FROM INVOICELINE WHERE invoiceid IN (SELECT invoiceid FROM INVOICE WHERE customerid IN (SELECT customerid FROM CUSTOMER WHERE firstname = 'Robert' AND lastname = 'Walter'))

• SQL Functions

In this section you will be using the Oracle system functions, as well as your own functions, to perform various actions against the database

3.1 System Defined Functions

Task – Create a function that returns the current time.

CREATE OR REPLACE FUNCTION getTime()

RETURNS TIMESTAMP AS $$

BEGIN

RETURN NOW();

END;

$$ LANGUAGE plpgsql

Task – create a function that returns the length of a mediatype from the mediatype table

CREATE OR REPLACE FUNCTION lenOf(mediatype VARCHAR)

RETURNS INTEGER AS $$

BEGIN

RETURN LENGTH(mediatype);

END;

$$ LANGUAGE plpgsql;

3.2 System Defined Aggregate Functions

Task – Create a function that returns the average total of all invoices

CREATE OR REPLACE FUNCTION

getAvgInvoice()

RETURNS TABLE (t NUMERIC) AS $$

BEGIN

RETURN QUERY SELECT AVG(total) FROM invoice;

END;

$$ LANGUAGE plpgsql;

SELECT getavginvoice();

Task – Create a function that returns the most expensive track

CREATE OR REPLACE FUNCTION

get\_max\_track()

RETURNS TABLE (name VARCHAR) AS $$

BEGIN

RETURN QUERY SELECT track.name FROM track WHERE unitprice > 1.98;

END;

$$ LANGUAGE plpgsql;

SELECT get\_max\_track();

3.3 User Defined Scalar Functions

Task – Create a function that returns the average price of invoiceline items in the invoiceline table

CREATE FUNCTION average\_price ()

RETURNS NUMERIC(10, 2) AS $av$

DECLARE

av NUMERIC(10, 2);

BEGIN

SELECT AVG(unitprice) FROM invoiceline INTO av;

RETURN av;

END;

$av$ LANGUAGE PLPGSQL;

3.4 User Defined Table Valued Functions

Task – Create a function that returns all employees who are born after 1968.

CREATE FUNCTION old\_people ()

RETURNS TABLE (

name VARCHAR,

bday TIMESTAMP

)

AS $v$

BEGIN

RETURN QUERY SELECT firstname, birthdate FROM employee WHERE birthdate > '1968-12-31';

END;

$v$ LANGUAGE PLPGSQL;

4.0 Stored Procedures

In this section you will be creating and executing stored procedures. You will be creating various types of stored procedures that take input and output parameters.

4.1 Basic Stored Procedure

Task – Create a stored procedure that selects the first and last names of all the employees.

CREATE FUNCTION employee\_selector()

RETURNS TABLE (

firstname VARCHAR,

lastname VARCHAR

)

AS $X$

BEGIN

RETURN QUERY SELECT E.firstname, E.lastname FROM employee E;

END;

$X$ LANGUAGE PLPGSQL;

4.2 Stored Procedure Input Parameters

Task – Create a stored procedure that updates the personal information of an employee.

CREATE FUNCTION personal\_info\_employee()

RETURNS void AS $$

BEGIN

UPDATE employee SET firstname = 'Bob' WHERE employeeid = 1;

END;

$$ LANGUAGE PLPGSQL;

Task – Create a stored procedure that returns the managers of an employee.

CREATE FUNCTION manger\_selector(reference INTEGER)

RETURNS TABLE (

firstname VARCHAR,

lastname VARCHAR

)

AS $Z$

BEGIN

RETURN QUERY SELECT E.firstname, E.lastname FROM employee E WHERE E.employeeid = reference;

END;

$Z$ LANGUAGE PLPGSQL;

4.3 Stored Procedure Output Parameters

Task – Create a stored procedure that returns the name and company of a customer.

CREATE FUNCTION company\_customer(custId INTEGER)

RETURNS TABLE (

company VARCHAR

firstname VARCHAR,

lastname VARCHAR,

)

AS $F$

BEGIN

RETURN QUERY SELECT C.firstname, C.lastname, C.company FROM customer C WHERE C.customerid = custId;

END;

$F$ LANGUAGE PLPGSQL;

5.0 Transactions

In this section you will be working with transactions. Transactions are usually nested within a stored procedure. You will also be working with handling errors in your SQL.

Task – Create a transaction that given a invoiceId will delete that invoice (There may be constraints that rely on this, find out how to resolve them).

CREATE FUNCTION invoice\_deleter(invId INTEGER)

RETURNS void AS $$

BEGIN

DELETE FROM invoiceline WHERE invoiceid = invId;

DELETE FROM invoice WHERE invoiceid = invId;

END;

$$ LANGUAGE PLPGSQL;

Task – Create a transaction nested within a stored procedure that inserts a new record in the Customer table

CREATE FUNCTION customer\_inserter()

RETURNS void AS $$

BEGIN

INSERT INTO customer VALUES (70, 'bob', 'spargle', 'Google', '852 West Lane', 'Venice', 'RM', 'Italy', '545', '587-4564', '454-5656', 4);

END;

$$ LANGUAGE PLPGSQL;

6.0 Triggers

In this section you will create various kinds of triggers that work when certain DML statements are executed on a table.

6.1 AFTER/FOR

Task - Create an after insert trigger on the employee table fired after a new record is inserted into the table.

CREATE TRIGGER insert\_post

AFTER INSERT ON employee

FOR EACH ROW

EXECUTE PROCEDURE suppress\_redundant\_updates\_trigger();

Task – Create an after update trigger on the album table that fires after a row is inserted in the table

CREATE TRIGGER trigger\_udpate

AFTER UPDATE ON album

FOR EACH ROW

EXECUTE PROCEDURE suppress\_redundant\_updates\_trigger();

Task – Create an after delete trigger on the customer table that fires after a row is deleted from the table.

CREATE TRIGGER delete\_post

AFTER DELETE ON customer

FOR EACH ROW

EXECUTE PROCEDURE suppress\_redundant\_updates\_trigger();

6.2 INSTEAD OF

Task – Create an instead of trigger that restricts the deletion of any invoice that is priced over 50 dollars.

7.0 JOINS

In this section you will be working with combing various tables through the use of joins. You will work with outer, inner, right, left, cross, and self joins.

7.1 INNER

Task – Create an inner join that joins customers and orders and specifies the name of the customer and the invoiceId. ANS:

SELECT customer, invoiceId

FROM customer

INNER JOIN invoice ON (customer.customerid = invoice.customerid);

7.2 OUTER

Task – Create an outer join that joins the customer and invoice table, specifying the CustomerId, firstname, lastname, invoiceId, and total.ANS:

SELECT customer.customerid, customer.firstname, customer.lastname, invoice.total, invoice.invoiceid FROM customer

LEFT JOIN invoice ON (customer.customerid = invoice.customerid)

UNION

SELECT customer.customerid, customer.firstname, customer.lastname, invoice.total, invoice.invoiceid FROM customer

RIGHT JOIN invoice ON (customer.customerid = invoice.customerid)

7.3 RIGHT

Task – Create a right join that joins album and artist specifying artist name and title.

SELECT album.artistid, album.title FROM album

RIGHT JOIN artist ON (album.artistid = album.artistid);

7.4 CROSS

Task – Create a cross join that joins album and artist and sorts by artist name in ascending order.

SELECT artist.name

FROM artist

CROSS JOIN album

ORDER BY name ASC;

7.5 SELF

SELECT a.reportsto

FROM employee a, employee b;