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 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, 'Britpop');

INSERT INTO genre (genreid, name) VALUES (27, 'K-Pop');

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, 'Gordon', 'Kevin', 'Intern', '1', '1996-03-07', '2018-08-15', '7050 3 Street NW', 'Calgary', 'AB', 'Canada', 'T2P 6S8', '+1 (403)760-5385','+1(403)534-9023', 'kevin@chinookcorp.com');

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

(10, 'Miller', 'Parker', 'Intern', '1', '1996-01-06', '2018-08-15', '3213 21 Ave', 'Calgary', 'AB', 'Canada', 'T2P 9B7', '+1 (403)443-9441','+1(780)639-0932', 'parker@chinookcorp.com');

Task – Insert two new records into Customer table

## 2.4 UPDATE

Task – Update Aaron Mitchell in Customer table to Robert Walter

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

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

UPDATE artist SET name = 'CCR' WHERE name = 'Creedence Clearwater Revival';

## 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 '01-06-2003' AND '01-03-2004';

## 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).

# 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 currentTime()

RETURNS refcursor as $$

DECLARE

curs refcursor;

BEGIN

OPEN curs for SELECT current\_time;

RETURN curs;

END;

$$ LANGUAGE plpgsql;

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

CREATE OR REPLACE FUNCTION length\_mediatype()

RETURNS refcursor as $$

DECLARE

curs refcursor;

BEGIN

OPEN curs for SELECT LENGTH(name), name FROM mediatype;

RETURN curs;

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 average\_invoice()

RETURNS refcursor AS $$

DECLARE

curs refcursor;

BEGIN

OPEN curs FOR SELECT avg(total) FROM invoice;

RETURN curs;

END;

$$ LANGUAGE plpgsql;

Task – Create a function that returns the most expensive track

CREATE OR REPLACE FUNCTION expensive\_track()

RETURNS refcursor AS $$

DECLARE

curs refcursor;

BEGIN

OPEN curs FOR SELECT MAX(unitprice), name from track;

RETURN curs;

END;

$$ LANGUAGE plpgsql;

## 3.3 User Defined Scalar Functions

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

CREATE OR REPLACE FUNCTION average\_invoiceline\_items()

RETURNS refcursor AS $$

DECLARE

curs refcursor;

BEGIN

OPEN curs FOR SELECT avg(unitprice) FROM invoiceline;

RETURN curs;

END;

$$ LANGUAGE plpgsql;

## 3.4 User Defined Table Valued Functions

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

CREATE OR REPLACE FUNCTION born\_after\_sixty\_eight()

RETURNS refcursor AS $$

DECLARE

curs refcursor;

BEGIN

OPEN curs FOR SELECT \* FROM employee WHERE birthdate > '01-01-1968';

RETURN curs;

END;

$$ 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 OR REPLACE FUNCTION employee\_names()

RETURNS refcursor AS $$

DECLARE

curs refcursor;

BEGIN

OPEN curs FOR SELECT lastname, firstname FROM employee;

RETURN curs;

END;

$$ LANGUAGE plpgsql;

## 4.2 Stored Procedure Input Parameters

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

CREATE OR REPLACE FUNCTION updated\_employee(employeeid INTEGER, lastname VARCHAR(20), firstname VARCHAR(20), title VARCHAR(30),

reportsto INTEGER, birthdate TIMESTAMP, hiredate TIMESTAMP, address VARCHAR(70), city VARCHAR(40), state VARCHAR(40),

country VARCHAR(40), postalcode VARCHAR(10), phone VARCHAR(24), fax VARCHAR(24), email VARCHAR(60))

RETURNS refcursor AS $$

DECLARE

curs refcursor;

BEGIN

OPEN curs FOR UPDATE customers

SET employeeid = 13,

lastname = 'Dixon',

firstname = 'Richard',

title = 'Regional Manager',

reportsto = null,

birthdate = '03-07-1970',

hiredate = '23-11-1990',

address = '129 Mulholland Dr',

city = 'Calgary',

state = 'AB',

country = 'Canada',

postalcode = 'T8G 8V0',

phone = '+1 (780)556-9909',

fax = '+1 (403)767-4976',

email = 'richyd@chinookcorp.com'

WHERE employeeid = employeeid;

RETURN curs;

END;

$$ LANGUAGE plpgsql;

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

CREATE OR REPLACE FUNCTION return\_manager(employeeid INTEGER)

RETURNS refcursor AS $$

DECLARE

curs refcursor;

BEGIN

OPEN curs FOR SELECT lastname, firstname FROM employee WHERE employeeid IN

(SELECT reportsto FROM employee WHERE employeeid = employeeid);

RETURN curs;

END;

$$ LANGUAGE plpgsql;

## 4.3 Stored Procedure Output Parameters

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

CREATE OR REPLACE FUNCTION return\_manager(customerid INTEGER)

RETURNS refcursor AS $$

DECLARE

curs refcursor;

BEGIN

OPEN curs FOR SELECT lastname, firstname, company FROM customer WHERE customerid = customerid;

RETURN curs;

END;

$$ 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).

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

CREATE OR REPLACE FUNCTION insert\_new\_customer(

c\_id INTEGER,

f\_name VARCHAR(40),

l\_name VARCHAR(20),

comp VARCHAR(80),

add VARCHAR(70),

cty VARCHAR(40),

st VARCHAR(40),

cntry VARCHAR(40),

zip VARCHAR(10),

numb VARCHAR(24),

fx VARCHAR(24),

mail VARCHAR(60),

s\_id INTEGER

)

RETURNS void AS $$

BEGIN

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

VALUES (c\_id, f\_name,l\_name, comp, add, cty, st, cntry, zip, numb, fx, mail, s\_id);

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 OR REPLACE FUNCTION insert\_trigger\_function()

RETURNS TRIGGER AS $$

BEGIN

IF(TG\_OP = 'INSERT') THEN

INSERT INTO employee (

old\_employeeid,

new\_employeeid,

old\_lastname,

new\_lastname,

old\_firstname,

new\_firstname,

old\_title,

new\_title,

old\_reportsto,

new\_reportsto,

old\_birthdate,

new\_birthdate,

old\_hiredate,

new\_hiredate,

old\_address,

new\_address,

old\_city,

new\_city,

old\_state,

new\_state,

old\_country,

new\_country,

old\_postalcode,

new\_postalcode,

old\_phone,

new\_phone,

old\_fax,

new\_fax,

old\_email,

new\_email

)VALUES(

null,

NEW.employeeid,

null,

NEW.lastname,

null,

NEW.firstname,

null,

NEW.title,

null,

NEW.reportsto,

null,

NEW.birthdate,

null,

NEW.hiredate,

null,

NEW.address,

null,

NEW.city,

null,

NEW.state,

null,

NEW.country,

null,

NEW.postalcode,

null,

NEW.phone,

null,

NEW.fax,

null,

NEW.email

);

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER insert\_trigger

BEFORE INSERT ON employee

FOR EACH ROW

EXECUTE PROCEDURE insert\_trigger\_function();

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

CREATE OR REPLACE FUNCTION update\_trigger\_function()

RETURNS TRIGGER AS $$

BEGIN

IF(TG\_OP = 'UPDATE') THEN

INSERT INTO album (

old\_albumid,

new\_albumid,

old\_title,

new\_title,

old\_artistid,

new\_artistid

)VALUES(

OLD.albumid,

NEW.albumid,

OLD.title,

NEW.title,

OLD.artistid,

NEW.artistid

);

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER update\_trigger

BEFORE UPDATE ON album

FOR EACH ROW

EXECUTE PROCEDURE update\_trigger\_function();

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

CREATE OR REPLACE FUNCTION delete\_trigger\_function()

RETURNS TRIGGER AS $$

BEGIN

IF(TG\_OP = 'DELETE') THEN

INSERT INTO customer(

old\_customerid,

new\_customerid,

old\_firstname,

new\_firstname,

old\_lastname,

new\_lastname,

old\_company,

new\_company,

old\_address,

new\_address,

old\_city,

new\_city,

old\_state,

new\_state,

old\_country,

new\_country,

old\_postalcode,

new\_postalcode,

old\_phone,

new\_phone,

old\_fax,

new\_fax,

old\_email,

new\_email,

old\_supportrepid,

new\_supportrepid

)VALUES(

OLD.customerid,

null,

OLD.firstname,

null,

OLD.lastname,

null,

OLD.company,

null,

OLD.address,

null,

OLD.city,

null,

OLD.state,

null,

OLD.country,

null,

OLD.postalcode,

null,

OLD.phone,

null,

OLD.fax,

null,

OLD.email,

null,

OLD.supportrepid,

null

);

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER delete\_triger

BEFORE DELETE ON customer

FOR EACH ROW

EXECUTE PROCEDURE delete\_trigger\_function();

## 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.

SELECT lastname, firstname, 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.

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

FROM customer

FULL OUTER 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 artist.name, album.title

FROM album

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

## 7.4 CROSS

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

SELECT \* FROM artist CROSS JOIN album

ORDER BY artist.name;

## 7.5 SELF

Task – Perform a self-join on the employee table, joining on the reportsto column.

SELECT \* FROM employee a, employee b

WHERE a.reportsto = b.reportsto;