**SQL Lab**

# Setting up Postgres Chinook

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

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

## 2.0 SQL Queries

In this section you will be performing various queries against the Postgres 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 is 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, city **from** customer **order** **by** city **asc**;

## 2.3 INSERT INTO

Task – Insert two new records into Genre table

insert into genre(genreid, name) values ('26', 'Oldies');

insert into genre(genreid, name) values (‘27’, ‘Indie Folk’);

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, 'Felix', 'Dominique', 'Software Engineer', 1, '1994-08-13 00:00:00', '2019-01-23 00:00:00', '2414 Scott St', 'Hollywood', 'FL', 'USA', 33020, 3057446928, **null**, 'dfeli014@fiu.edu')

**insert** **into** employee(employeeid, lastname, firstname, title, reportsto, birthdate, hiredate, address, city,state,country, postalcode, phone, fax, email) **values** (10, 'Horton', 'Tim', 'Software Engineer', 1, '1996-09-29 00:00:00', '2019-01-23 00:00:00', '14525 Prism Circle', 'Tampa', 'FL', 'USA', 33163, 8139043212, **null**, 'timhto2@gmail.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**(60, 'Jacob', 'Gonçalves', 'Embraer - Empresa Brasileira de Aeronáutica S.A.', 'Av. Brigadeiro Faria Lima, 2170', 'São José dos Campos', 'SP', 'Brazil', '12227-000', '+55 (12) 3923-5555', '+55 (12) 3923-5566', 'jg@embraer.com.br', 3);

**INSERT** **INTO** customer (customerid, firstname, lastname, company, address, city, state, country, postalcode, phone, fax, email, supportrepid) **VALUES**(61, 'Ashley', 'Köhler', **NULL**, 'Theodor-Heuss-Straße 34', 'Stuttgart', **NULL**, 'Germany', '70174', '+49 0711 2842222', **NULL**, 'ashleyk@surfeu.de', 5);

## 2.4 UPDATE

Task – Update Aaron Mitchell in Customer table to Robert Walter

**UPDATE** customer **SET** firstname='Robert', lastname='Walter', company=**NULL**, address='696 Osborne Street', city='Winnipeg', state='MB', country='Canada', postalcode='R3L 2B9', phone='+1 (204) 452-6452', fax=**NULL**, email='aaronmitchell@yahoo.ca', supportrepid=4 **WHERE** customerid=32;

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

**UPDATE** public.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).

**ALTER** **TABLE** invoice **DROP** **CONSTRAINT** fk\_invoicecustomerid;

**delete** **from** customer **where** customerid=32;

# 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** **function** currenttime() **returns** **timestamptz**

**as** 'select NOW();'

**language** **sql**

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

**create** **function** mediatypelen(**text**) **returns** **integer** **as** **$$**

**select** **char\_length**($1);

**$$** **language** **sql**;

## 3.2 System Defined Aggregate Functions

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

**create** **function** totalInvoices() **returns** **numeric**(10,10) **as** **$$**

**select** **avg**(total) **from** invoice;

**$$** **language** **sql**;

Task – Create a function that returns the most expensive track

**create** **function** mostExpensiveTrack() **returns** **numeric**(10,10) **as** **$$**

**select** **max**(unitprice) **from** track;

**$$** **language** **sql**;

## 3.3 User Defined Scalar Functions

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

**create** **function** avgInvoiceLinePrice() **returns** **numeric**(10,10) **as** **$$**

**select** **avg**(unitprice) **from** invoiceline;

**$$** **language** **sql**;

## 3.4 User Defined Table Valued Functions

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

**create** **function** employeesAfter() **returns** **table**(f1 **integer**, f2 **text**, f3 **text**, f4 **timestamp**)

**as** **$$** **select** employeeid, firstname, lastname, birthdate **from** employee **where** birthdate>'1968-12-31';

**$$** **language** **sql**;

# 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** namesOfEmployees() **returns** **TABLE**(firstname **text**, lastname **text**) **as**

**$$** **select** firstname, lastname **from** employee

**$$** **language** **sql**;

## 4.2 Stored Procedure Input Parameters

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

**create** **function** changeInfo(fname **text**, lname **text**, address **text**, city **text**,state **text**,country **text**, postalcode **integer**, phone **integer**, fax **integer**, email **text**)

**returns** **void** **as**

**$$** **update** employee **set** firstname=$1, lastname=$2, address=$3, city=$4, state=$5, country=$6, postalcode=$7, phone=$8, fax=$9, email=$10;

**$$** **language** **sql**;

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

**create** **function** whichManager(empid **integer**) **returns** **integer** **as**

**$$** **select** reportsto **from** employee **where** employeeid=$1;

**$$** **language** **sql**;

## 4.3 Stored Procedure Output Parameters

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

**create** **function** custInfo(custid **integer**) **returns** **table**(firstname **text**, lastname **text**, company **text**) **as**

**$$** **select** firstname, lastname, company **from** customer;

**$$** **language** **sql**;

# 5.0 Transactions

In this section you will be working with transactions. Transactions are usually nested within a stored procedure.

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).

**begin**

**alter** **table** invoice **drop** **constraint** fk\_invoicecustomerid;

**alter** **table** invoiceline **drop** **constraint** fk\_invoicelineinvoiceid;

**delete** **from** invoice **where** invoiceid=2;

**alter** **table** invoice **add** **constraint** fk\_invoicecustomerid

**foreign** **key** (customerid)

**references** customer(customerid);

**alter** **table** invoiceline **add** **constraint** fk\_invoicelineinvoiceid

**foreign** **key** (invoiceid)

**references** invoice(invoiceid);

**end;**

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

**create** **or** **replace** **function** insertCustomer(idnum **integer**, fname **text**, lname **text**, comp **text**, addr **text**, city **text**, state **text**, country **text**, postalcode **integer**, phone **integer**,fax **integer**, email **text**, supprepid **integer**)

**returns** **void** **as** **$$**

**begin**

**insert** **into** customer (customerid, firstname, lastname, company, address, city, state, country, postalcode, phone, fax, email, supportrepid)

**values**(idnum, fname, lname, comp, addr, city, state, country, postalcode, phone, fax, email, supprepid);

**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** after\_insert

**after** **insert** **on** employee

**for** **each** **row**

**execute** **procedure** insertTrig();

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

**create** **trigger** after\_update\_album

**after** **update** **on** album

**for** **each** **row**

**execute** **procedure** afterTrig();

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

**create** **trigger** after\_delete\_customer

**after** **update** **on** customer

**for** **each** **row**

**execute** **procedure** afterDelete();

# 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** invoiceid **as** "invoiceid", firstname **as** "fname", lastname **as** "lname" **from** customer C

**Inner** **join** invoice I **on** c.customerid = i.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** c.customerid **as** "customerid", firstname **as** "firstname", lastname **as** "lastname", invoiceid **as** "invoiceid", total **as** "total"

**from** invoice I **full** **join** customer C **on** C.customerid = I.customerid;

## 7.3 RIGHT

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

**select** **name** **as** "artistname", title **as** "title" **from** album A

**right** **join** artist art **on** a.artistid = art.artistid;

## 7.4 CROSS

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

**select** **name** **as** "artistname" **from** album a

**cross** **join** artist **order** **by** **name** **asc**;

## 7.5 SELF

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

**select** e.firstname, e.lastname, e.reportsto **from** employee e **inner** **join** employee m **on** **true**;