Week 5 slides (pre-recorded lecture) have reference to the numbering scheme followed here, e.g., SQL1, SQL2. Please use **chinook database** to execute these examples and try your own queries!

SQL1: LIMIT, ORDER BY and OFFSET

sqlite> SELECT trackid, name, bytes FROM tracks **ORDER BY** bytes **DESC LIMIT** 10;

sqlite> SELECT TrackId, Name, Composer FROM tracks **ORDER BY** Composer **LIMIT** 10;

sqlite> SELECT trackId, name FROM tracks **LIMIT 10 OFFSET 10**;

SQL2: DISTINCT

(If you use one column, SQLite uses values in that column to evaluate the duplicate. In case you use multiple columns, SQLite uses the combination of values in these columns to evaluate the duplicate)

sqlite> SELECT DISTINCT city FROM customers ORDER BY city;

sqlite> SELECT DISTINCT city, country FROM customers ORDER BY country;

SQL3: Calculated field

sqlite> select TrackId, Name, UnitPrice, **UnitPrice*2 as "New price"** from tracks limit 10;

SQL4: NOT

sqlite> select TrackId, Name, UnitPrice, UnitPrice*2 as "New price" from tracks WHERE trackid < 20 AND **NOT** Name like "F%" limit 10;

SQL5: WHERE + AND

(Where operator and its use with comparison (=, <>, etc.) and logical operators (AND, OR, LIKE, etc.)

sqlite> SELECT name, milliseconds, bytes, albumid FROM tracks WHERE albumid = 1 **AND** milliseconds > 250000;

SQL6: BETWEEN

sqlite> SELECT InvoiceId, BillingAddress, Total FROM invoices WHERE Total **BETWEEN** 14.91 and 18.86 ORDER BY Total;

SQL7: IN

sqlite> SELECT name, albumid, mediatypeid FROM tracks WHERE mediatypeid **IN** (2, 3) ORDER BY albumid LIMIT 5;

SQL8: LIKE

sqlite> SELECT name, albumid, composer FROM tracks WHERE composer **LIKE** '**%**Smith**%**' ORDER BY albumid LIMIT 15;

SQL9: NULL

sqlite> SELECT Name, Composer FROM tracks WHERE Composer **IS NULL** ORDER BY Name LIMIT 25;

SQL10: SQL aggregate functions

sqlite> SELECT **COUNT**(composer), **COUNT** (distinct composer) FROM tracks;

SQL11: GROUP BY

sqlite> SELECT albumid, COUNT(trackid) FROM tracks **GROUP BY** albumid ORDER BY COUNT(trackid) DESC LIMIT 10;

SQL12: GROUP BY with HAVING

sqlite> SELECT albumid, COUNT(trackid) FROM tracks **GROUP BY** albumid HAVING **COUNT**(trackid) > 25;

SQL13: SUBQUERIES

sqlite> SELECT trackid, name, albumid FROM tracks WHERE albumid **= (SELECT albumid FROM albums WHERE title = 'Let There Be Rock')**;

sqlite> SELECT customerid, firstname, lastname FROM customers WHERE supportrepid **IN (SELECT employeeid FROM employees WHERE country = 'Canada')**;

SQL14: For JOINS - First create two tables and insert records

sqlite> CREATE TABLE Staff (

staffID INTEGER PRIMARY KEY,

staffName TEXT (50));

sqlite> CREATE TABLE Student (

studentID INTEGER PRIMARY KEY,

studentName TEXT (50),

staffID INTEGER -- acts as student's personal tutor

);

sqlite>

insert into Staff values (1, "S1");

insert into Staff values (2, "S2");

insert into Staff values (3, "S3");

insert into Student values (1, "Student1", 1);

insert into Student values (2, "Student2", 3);

insert into Student values (3, "Student3", 3);

insert into Student values (4, "Student4", 3);

insert into Student values (5, "Student5", 1);

SQL14-A: Inner Join

sqlite> select staff.staffID, staffName, studentID, studentName, student.staffID from staff, student **WHERE** staff.staffID=student.staffID;

sqlite> select staff.staffID, staffName, studentID, studentName, student.staffID from staff **INNER JOIN** student **ON** student.staffID=staff.staffID;

SQL14-B: Left Join

sqlite> select staff.staffID, staffName, studentID, studentName, student.staffID from staff **LEFT JOIN** student **ON** student.staffID=staff.staffID;

SQL14-C: Cross Join

sqlite> select staff.staffID, staffName, studentID, studentName, student.staffID from staff **CROSS JOIN** student;

SQL15: Self Join

sqlite> SELECT m.firstname || ' ' || m.lastname AS 'Manager', e.firstname || ' ' || e.lastname AS 'Direct report' FROM employees **e INNER JOIN** employees **m** ON **m.employeeid = e.reportsto** ORDER BY manager;

/End