Assignment Number:	INB210/INN210.1
Assignment Name:	SQL Assignment
Weighting:	20%. • *Each of the tasks* of this assignment *is worth an equal amount*. In general, part marks will be given for a partly correct answer, for each of the tasks.
Due Date:	Friday, 13th September 2013 • Rather than leaving it to the last minute, you are strongly encouraged to start working on this assignment at a much earlier stage. The relevant material is covered in Lectures 4 through 6, and in the corresponding practicals. • Students who have little or no Web access at home, and who are *not* normally on campus on the day the assignment is due, are encouraged to submit their work when they attend classes during the preceding 7 days. (There will not be any last minute changes to the assignment requirements.)
Items to be Submitted:	The eleven *.sql files described in the main body of this assignment. *A printed copy is not required, and will be ignored if submitted.*
How to be Submitted:	The assignment must be submitted using the BlackBoard – see page 4.

Before submitting your assignment, check that you have met all the requirements listed below. You don't want to lose marks for things that you might think are unimportant, but that others view differently.

Do's and Don'ts

Academic Honesty

(a) While practicals may encourage you to work with fellow students, INB210/INN210 assignments are *not* group work. When you submit your assignment, you will tick a box that begins: "I confirm that that this work represents my individual effort".

While you may discuss aspects of an assignment with other students and with academic staff, it is unethical to copy part or all of the work of another person and claim it as your own. QUT takes academic honesty very seriously. **In past semesters, some people have ignored these warnings and have received academic penalties, sometimes leading to overall grades of "failure".**

A description of the full range of penalties may be seen by following the **Academic misconduct** link in the **Assessment** page on Blackboard (http://www.student.qut.edu.au/studying/assessment/academic-misconduct).

As well as wishing to avoid such penalties, there is another good reason for doing assignments on your own. The final exam includes questions that test for knowledge gained by doing the assignments. People who take short-cuts when doing assignments find that they cannot answer such exam questions, despite cramming before the exam.

Be smart, don't learn this lesson the hard way!

The Assignment

(b) In general, the lower-numbered tasks are *easier* than the higher-numbered ones, but this may not always true. If you get stuck on one query, leave it for a while and try another one.

(c) *All the files you submit must be text files, not files created by a Word Processing program*, i.e. *no* .docx files. If your files cannot be read sensibly by a text-based program, such as SqlFire or NotePad, then a mark of zero will be given.

(d) We expect your answer files (.sql files) to *contain only valid SQL statements* (according to the SQL:2008 standard). *Do not put anything else in these files*. For example, *do not include unnecessary (non-SQL) words*, such as "QUERY 1". If we automatically execute the file, the SQL software will assume that you do not understand SQL, and so will give you zero.

(e) You can use another DBMS to write your queries. But make sure that the query runs on SqlFire. Markers would execute your queries on SqlFire to make sure that it works correctly. They would not be engaged in modifying your queries to make them compatible with SqlFire.

If you don't use SqlFire, then use a text editor (such as Notepad or Notepad++) to create the .sql files. As stated above, *do not* use a word processor, such as Microsoft Word.

(f) Don't select more columns than are asked for, in each task. E.g. don't write "select *" when only a few specific columns are asked for.

(g) Don't use anything that is not part of SQL:2008 – i.e. not in the Lecture Notes – even if it is allowed by some particular DBMS. For example, Microsoft Access allows square brackets around column names, but *SQL:2008 does not*.

(h) Don't make any of the **Frequent Mistakes in Writing SQL**, listed in Chapter 6 of the Lecture Notes. These kinds of mistakes have been collected from past assignments. W*hen you think you've finished your answer, it's a good time to see if you can make it better, by making it simpler. But not too simple, see point (l) below.*

(i) Putting a semicolon (;) at the end of each query is optional. Please yourself.

NB: You'll probably find it helpful to use the actual contents of the example VideoStore database to help you develop your answers. But, when you do, there are two things to be careful about.

(j) When writing your SQL queries, *you must not assume that particular rows are present (or absent)* in the database. That is, your SQL queries *must still give appropriate results, even if all the current rows were deleted and a completely different set of rows inserted*.

As a simple example of this, if you were asked to retrieve information related to Cameron Diaz, it would be *wrong to write a query assuming that her StarId was 181*. With different rows (or an update to the database), it might be a *different value*! So, to do this properly, you would have to write a query that looks for her name, rather than a particular StarId.

In other words, it is very important to *write queries that give results conforming to the query as stated in English*, I.e. results that are appropriate even when the rows change. This rule applies to this assignment, the Access assignments, the final exam, and to the real world!

(k) The data in the example VideoStore database can be used to help you develop your answers, *but there is no requirement to use the given rows in that database, as supplied to you*. In particular, there is no guarantee that the data in that database will be sufficient to test whether your answers always work correctly or not.

Specifically, there is no guarantee that each query will actually retrieve *any* rows. I.e. some might not find any matching rows. This does not necessarily mean that your answer is wrong. For example, think about the first query being used with a database in which *everyone* in Toowong had a home phone number.

What would you do in such a case? Well, remember that there's nothing special about the actual rows in this database. You can always add new ones, change existing ones, or delete them, to check that your query retrieves the data that you think it should. So, if there were no customers with just a mobile phone, it would be sensible for you to insert some!

(l) Each query will be given a raw ("unweighted") mark on a scale of 0..4, with each value having the following meaning:

4	completely correct;
3	one thing wrong;
2	two or more different things wrong, but other important parts correct;
1	answer is largely wrong or incomplete, but some (partially-correct) attempt has been made;
0	answer not attempted or is completely wrong.

The total raw mark will be converted proportionately into a corresponding weighted mark. For example, a raw mark of 22 out of 44 will become 10% out of 20%.

(m) A penalty will automatically be deducted for late assignments. Standard policy is:

Number of Days Late	**Penalty**
1 day	10%
2 days	20%
3-4 days	30%
5-7 days	40%
8-10 days	50%
More than 10 days	100%

Note that "days late" includes weekends.

Using Blackboard to Submit Your Assignment

You must submit your assignment using Blackboard. Note:

(a) Put all eleven queries (e.g., 1.sql, 2.sql....11.sql) inside a zip file. Use the filename **SqlAssign.zip**. Do not include any other files in your zip file, because the automated processing will ignore them. Do not try to use any other type of compressed file, such as 7z, arc, etc.

(b) In general, problems such as "couldn't figure out how to submit via Blackboard" or "couldn't access Blackboard from home" *will **not** be accepted as reasons for getting an extension*, i.e. avoiding a penalty for a late submission. Because of this, you are strongly encouraged to:

i. *submit the assignment before the due date*, rather than waiting until the last day; and

ii. *have a **practice go*** at submitting the assignment in Blackboard, well before the due date, so that you know how it works in advance. When doing this, you can submit any small zip file that you have created. Its content can be complete junk. If you wish, you can do this in a prac, so that your staff member can assist you with any problems.

(c) In INB210/INN210, you are allowed to *submit the assignment more than once, on or before the due date*. So, if you've submitted the assignment early and then suddenly think of a better answer in time, you can resubmit the assignment. But *do not resubmit after the due date*. Doing so would give you a late penalty, as well as making it likely that the earlier version of your assignment will be marked.

NB: if you resubmit, *you must resubmit the **whole assignment***, not just the bits that have changed. Whenever you resubmit, *your entire earlier submission is deleted* from the system. For example, if you resubmit *only* the file 3.sql, it will appear that this is the only part of the assignment you've done, so you'll get *no marks* for the other parts of the assignment.

(d) General information about submitting assignments using Blackboard is available here. http://www.els.qut.edu.au/blendedlearning/blackboard/students/tipsheets/assignments.jsp

(e) A general announcement that the marking has been completed will be placed on Blackboard after about two weeks of the assignment due date.

The Database for this Assignment

The VideoStore database contains the following seven relations ("tables"):

Movies (<u>MovieNum</u>, Title, YearReleased)
Stars (<u>StarId</u>, GivenName, FamilyName)
ActsIn (<u>StarId, MovieNum</u>)
Copies (<u>BarCode</u>, MovieNum, Format, RentalCode)
RentalRates (<u>RentalCode</u>, RentalPeriod, Rate)
Customers (<u>CustomerNum</u>, GivenName, FamilyName, Address, HomePhoneNum, MobilePhoneNum)
Rentals (<u>CustomerNum, BarCode</u>, DateDue, DateReturned)

The meaning of most of these tables and their attributes ("column names") should be fairly obvious. Here is a brief description of some aspects that may be less obvious:

- The only columns that allow nulls are HomePhoneNum, MobilePhoneNum, and DateReturned.
- BarCode is a unique identifier for each rental item in the store. That is, every item has a different bar code.
- Format is either "DVD" or "Blu-ray".
- Some customers only have home phone numbers. Some only have mobile phone numbers. Some have neither.
- When a customer borrows a video, a row is inserted into the Rentals table, with DateDue set to the date by when the item must be returned, and DateReturned set to null. When the customer returns the video, DateReturned is set to that date. This lets us see which rentals are current, and which are in the past. (On the rare occasions when a customer rents the same physical item on different days, the older row is removed first.)

This is a deliberately cut-down database structure, compared to what is required in a real video store, but it contains some essential parts. We will not use every column, in this assignment.

As an example of this VideoStore database, a SqlFire-format database – named VideoStore.sfdb – is available on Blackboard. The data in this database is just a small sample of the data that a real video store would have.

The Tasks

You may find it helpful to draw yourself a Relational Map of the above database, at the start, to make it easier to do tasks in this assignment.

Hint (Tasks 1 to 3): joins and subqueries are not covered until Lectures 5 and 6, and so should not be used for any queries in this part of the assignment.

1. Create a textfile, named 1.sql, containing a single SQL statement that does the following.

   List all the details (i.e. all columns) of Toowong customers who don't have a home phone number. Note that "Toowong" may occur anywhere in their address, rather than just at the end.

2. Create a textfile, named 2.sql, containing a single SQL statement that does the following.

   List the year-released and title (only) of each movie that was released after the end of 2004 but before the end of 2010. List this information in ascending order of year-released, and then by alphabetical order of title whenever the year-released is the same.

3. Create a textfile, named 3.sql, containing *a single SQL statement* that answers all the following questions. Do *not* write four different SQL statements.

   How many movies are recorded in the database? (Note: this is meant to be a simple question. Do not be concerned about the fact that some movies may have the same title, and do not be concerned about the number of copies of each. I.e. ignore such things.) Of all these movies, what is the oldest YearReleased? And, the most recent? And, how many *different* YearReleased values are there in the database?

Hint (Tasks 4 -7): subqueries are not covered until Lecture 6, and so are not needed for any queries in this part of the assignment.

4. Create a textfile, named 4.sql, containing a single SQL statement that does the following.

   For each year in the database, list the year and the total number of movies that were released in that year, showing these totals in decreasing order. That is, the year(s) with the largest number of movies appear first. If some years have the same number of movies, show these in increasing order of year. E.g. for some database – *not* the example one on Blackboard – we might have:

   ```
   2008    9
   1995    7
   1999    7
   2002    3
   ```

   This example illustrates the requirements stated above. The output is sorted by the right-hand column (in decreasing order), and when two output rows have the same value (7), those rows are sorted by their year (in increasing order).

5. Create a textfile, named 5.sql, containing a single SQL statement that does the following.

   List the titles and year released of all movies that Cameron Diaz is in. (Reminder: in your query, don't assume that you already know her StarId – see point (k) on page 2 of this assignment.)

6. Create a textfile, named 6.sql, containing a single SQL statement that answers the following question.

   What are the given and family names of stars who are in four or more movies, and how many movies are they in?

7. Create a textfile, named 7.sql, containing a single SQL statement that does the following.

   This query is about movies that have been made *more than once*, i.e. released in different years. For each pair of such movies, display one row showing the title and the years, e.g.

   ```
   The Day The Earth Stood Still   1951    2008
   The Nutty Professor            1963    1996
   The Nutty Professor            1963    2008
   The Nutty Professor            1996    2008
   The Pink Panther               1964    2006
   etc.
   ```

   NB:
   (a) The output is to be sorted by the title, and then by the left-most year value, and then by the right-most year value, as shown by the example rows above.

   (b) When a movie has been made twice, do not show it twice, with the years simply swapped around.
   E.g. don't show both of these rows:

   ```
   The Day The Earth Stood Still   1951    2008
   The Day The Earth Stood Still   2008    1951
   ```

   (c) When a movie has been made three times, it should appear as three rows -- see The Nutty Professor above -- because there are three possible pairs, i.e. three ways of choosing two values from three years.

Hint: we're not trying to summarise data in this query, so trying to use "group by" is a bad approach here.

Hint (Tasks 8 -11): use subqueries to complete these tasks.

8. Create a textfile, named 8.sql, containing a single SQL statement that answers the following question.

   Which movie titles are *only* available in DVD format, i.e. not Blu-ray? For each such movie, list both the title and the year released.

   Note that some movies are available for rental in *both* DVD and Blu-ray formats, while the others are available in *only one* of these. I.e. some are available only in DVD format, some are available only in Blu-ray format, and some are available in both.

9. Create a textfile, named 9.sql, containing a single SQL statement that answers the following question.

   Which movie copies have the lowest rental rate? For simplicity, you may assume that there is only one row in the RentalRates table that has the lowest rate, but do not assume that you know the specific values in that row. I.e., do not assume that you know that row's RentalCode, RentalPeriod or Rate.

   Your answer must list the title, year released and format of those copies with the lowest rates, but not show the same combination of title, year released and format more than once. Use of ORDER BY clause is prohibited.

10. Create a textfile, named 10.sql, containing a single SQL statement that answers the following question.

    What is the title and year released of the *latest* Eddie Murphy movie? (Reminder: don’t assume that *you* already know the year – see point (k) on page 2 of this assignment.) Use of ORDER BY clause is prohibited.

11. Create a textfile, named 11.sql, containing a single SQL statement that answers the following question.

    Which customers have the largest number of copies rented at present? Your answer must show the given and family names, and address, of each such customer. (Reminder: don’t assume that *you* already know the value of this largest number – see point (k) on page 2 of this assignment.) Use of ORDER BY clause is prohibited.

Reminder: before submitting your assignment, check that you have met all the requirements listed under the **Do’s and Don’ts** listed above, including avoiding the **Frequent Mistakes in Writing SQL** listed in Chapter 6 of the Lecture Notes. You don’t want to lose marks for things that you think are unimportant, but that the marker views differently.