Logical Modelling and Normalisation – Bex SHINRL1

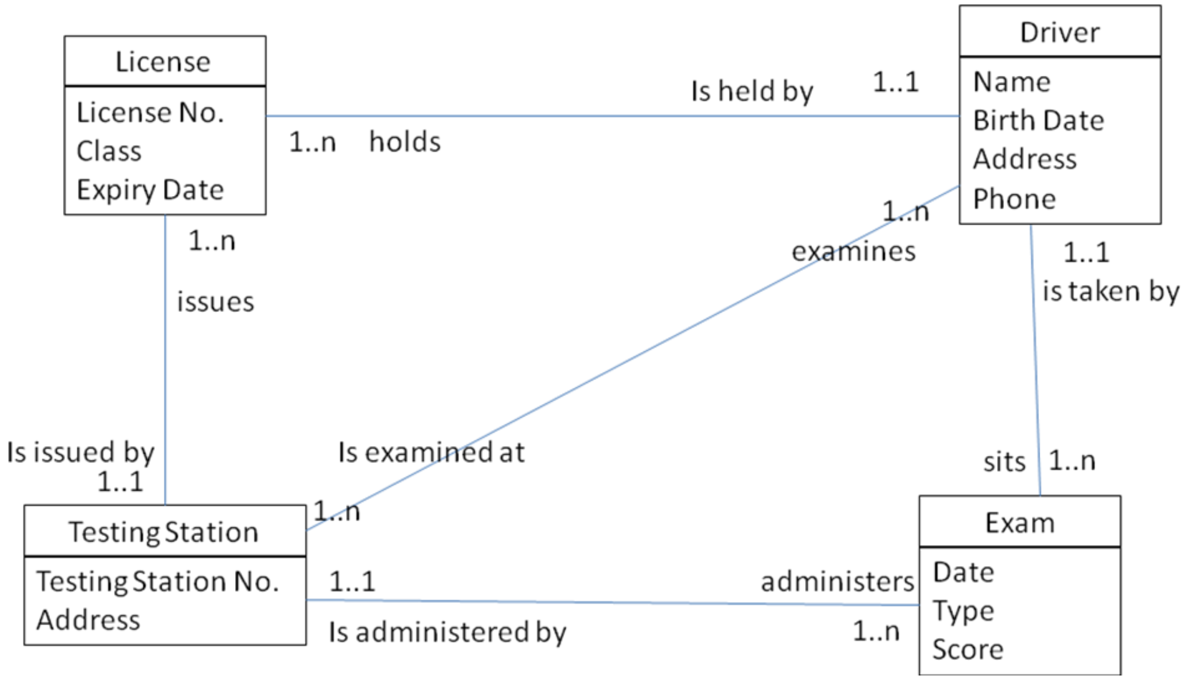
This is a checkpoint. You may work in pairs. Due Friday Week 12 (25th Oct), 5.00 pm

1. Convert the entity-relationship diagram below into a relational schema (logical model for a relational DBMS).
    i. List any additional entities you feel this model should have (but won't be used in this exercise)?

    DriverID – the structure assumes that there can only be on of each driver.
    ExamID – the structure assumes one exam per day

- My structure has allowed for multiple drivers to sit the same exam

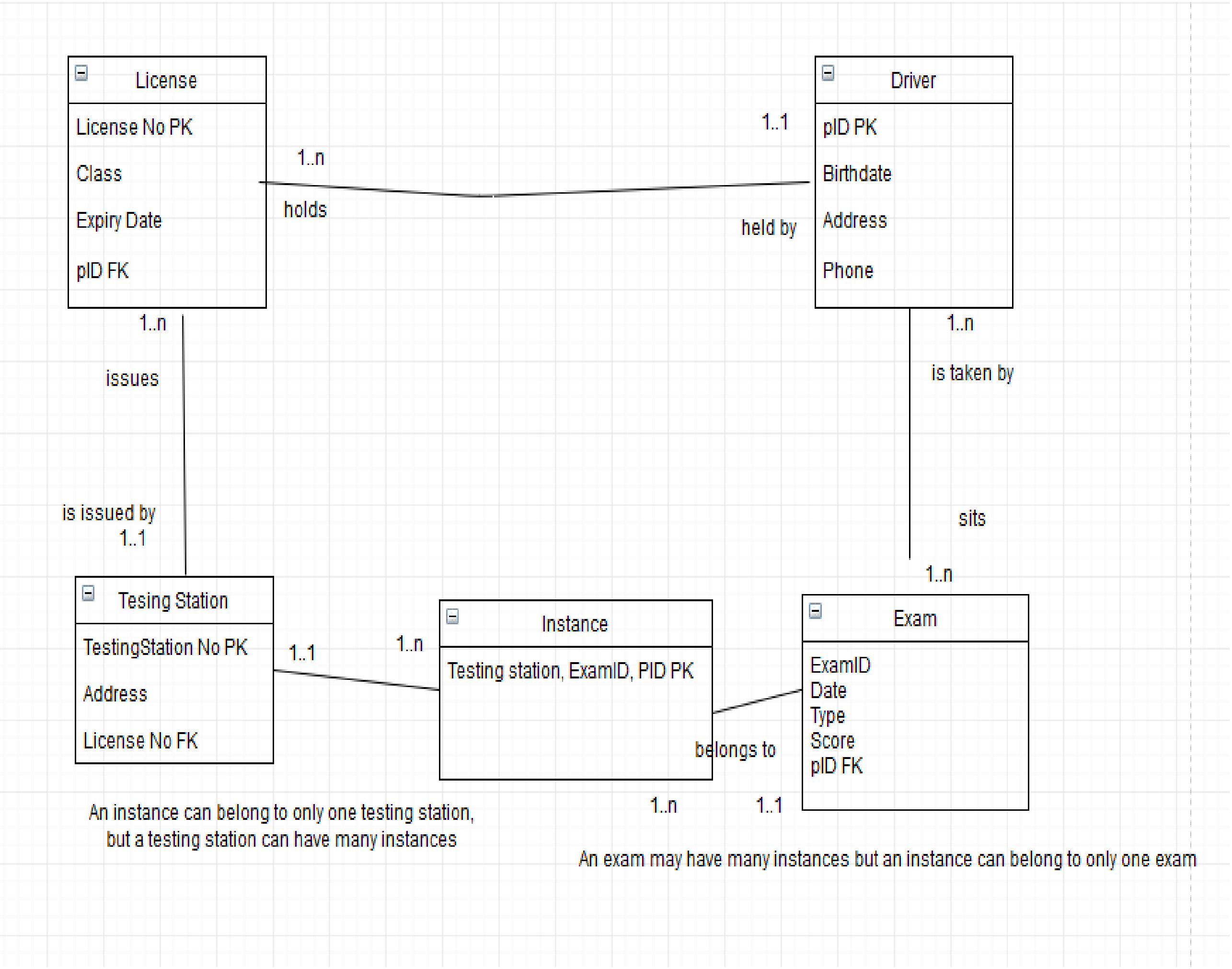
    ii. Represent your logical model using a UML-style diagram or *tableName(column, column, column….)* notation, as you prefer. Be sure to clearly indicate all primary and foreign keys.
    iii. Include a table which, for each column in your database, states the SQL Server data type and gives a brief description of the purpose of the field (i.e. an abridged *data dictionary*).



Note that this ERD contains a *ternary* relationship – that is, a relationship involving three entities. We did not discuss specifically how to translate higher-order relationships into relational schemata. However, you should be able to generalise from the treatment of a binary many-to-many relationship. Hint: A composite primary key may contain any number of fields.

Updated ERD

The following diagram demonstrates my solution of an updated structure where the ternary relationship has been translated into a one to many. This structure allows multiple exams to be sat on the same day, and for people to hold one or many licenses. I have added fields such as ExamID and pID to help solve preexisting problems, such as only one John Doe being allowed a license, or one exam to be sat only once per day.



An instance can belong to only one testing station, but a testing station can have many instances

An exam may have many instances but an instance can belong to only one exam

Data Dictionary

Column Name	**Type**	**Description**	**Table**
License No	Varchar or Int	Number that individually identifies driver string of numbers and/or letters thatindividually identifies license **PK**	License
Class	varchar	Type of license e.g learners, motorcycle etc	License
Expiry Date	date	Date which license is no longer valid	License
pID	int	Number that individually identifies driver. **FK**	License
Testing Station No	int	Number that indentifies station. **PK**	Testing Station
Address	varchar	Address at which station resides	Testing Station
License No	varchar	string of numbers and/or letters thatindividually identifies license, **FK**	Testing Station
Testing station, ExamID, PID	Composite Key – varchar int, int	**CK** that records the instance of a driver taking an exam	Instance
ExamID	ExamID	Individualy identifies exam **PK**	Exam
Date	Date	Date of exam	Exam
type	varchar	License type exam awards	Exam
score	int	Result of exam	Exam
pID	Int	Number that individually identifies driver. **FK**	Exam
pID	Int	Number that individually identifies driver. **PK**	Driver
Birthdate	date	Date driver was born	Driver
Address	Varchar	Address driver resides at	Driver
Phone	varchar	Number driver can be reached at	Driver

2. The schema below represents a database for storing information about the plays showing in South Island theatres. It is not normalised. Convert it to 3rd Normal Form. Carefully illustrate the process you follow (there should be three steps: UNF -> 1NF; 1NF->2NF; 2NF->3NF)

Theatre:

- Theatre Name
- Theatre Location
- No of Seats
- Year Founded
- Manager
- Manager Contact Number
- Play 1
    - Name, Genre, # performances, Author, Author address
- Play 2
    - Name, Genre, # performances, Author, Author address
- Play 3
    - Name, Genre, # performances, Author, Author address

First step: get rid of repetition:

First normal form – no multiple attributes! – take duplicate info and put it in a new table with the same primary key – as seen in play. If a field is a primary key it's okay to repeat it. This is not the case for all other fields.

- Make a Play Table instead of 3 different Play tables.
- **Play**(Name, Genre, # performances, Author (this will be pID))
- Author and Manager are essentially the same thing – let's make a Person table with a *role* column
- **Person**(pID, Name, Role, Contact Number, Address)
- **Theatre**(Name, Location, # seats, year founded, manager)

2nf – no partial dependency

- Oh no, we don't know what theatre has had what plays performed. Let's also add a foreign key to the Theatre table

- Play seems all good. All of the fields can be determined by the name of the pay

- Person is also good.

- Theatre – what if two theatres have the same name? Let's chuck Location into the key and make it a composite.

- **Play**(Name, Author, Genre, # performances)

- **Person**(pID, Name, Role, Contact Number, Address)

- **Theatre**([Name,Location], # seats, year founded, manager, play)

3NF

"no non-prime attribute depends on other non-prime attributes. All the non-prime attributes must depend only on the candidate keys."

This is where we deal with update/delete anomalies

Play seems all good. If genre goes, we still know everything else, same with # performances.

Person – Role? I think it's okay…

Theatre I also believe is okay.

I am struggling to differentiate steps from 2NF and 3NF. This is my attempt at normalizing the data set to 3NF

- **Play**(Name, Author Genre, # performances)

- **Person**(pID, Name, Role, Contact Number, Address)

- **Theatre**([Name Location], # seats, year founded, manager, play)