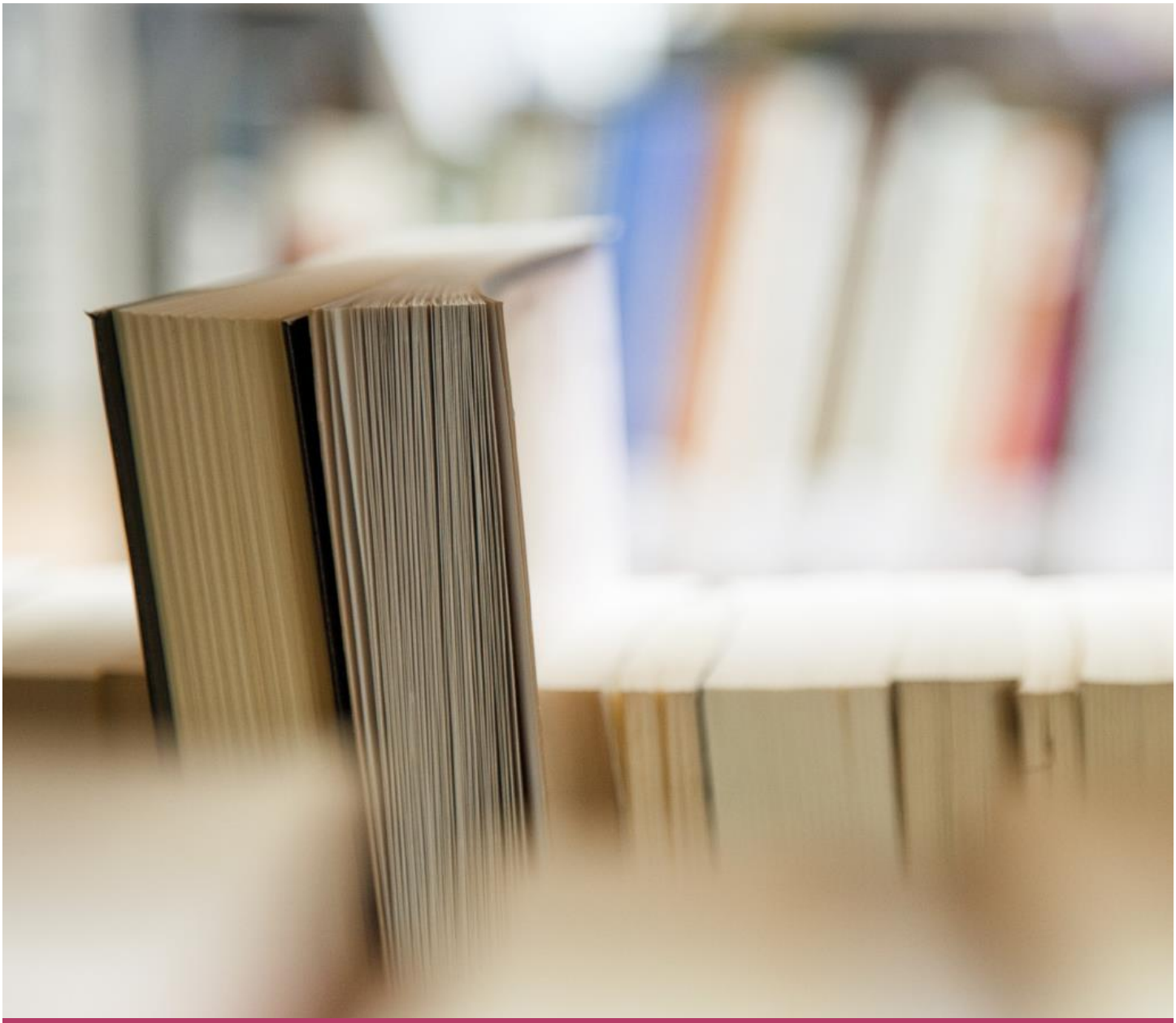


3612ICT Report

Elizabeth King s2800233

7/20/15

Database Administration

QUESTION 26. GIVEN A RELATIONAL DATABASE SCHEMA

Course (CourseNumber, Topic)

Student (StudentId, StudentName, StudentStreetAddress, Suburb)

Enrollment(CourseNumber, StudentId)

1.(1 mark) Write the following query in SQL "Retrieve the names of students who are enrolled in at least one course with topics starting with INTRODUCTION"

```
SELECT studentname FROM student s, Enrollment e, Course c
WHERE s.StudentId=e.StudentId AND
c.coursenumber=e.coursenumber AND LOWER(c.topic) LIKE
'%introduction%';
```

```
ij(CONNECTION1)> SELECT studentname FROM student s, Enrollment e, Course c WHERE
 s.StudentId=e.StudentId AND c.coursenumber=e.coursenumber AND LOWER(c.topic) LI
KE '%introduction%';
STUDENTNAME
--------------------
Orange
Penny
Penny

3 rows selected
```

To show this is working I have detailed other select statements I had used to validate my answer.

```
SELECT * FROM student s, Enrollment e, Course c

WHERE s.StudentId=e.StudentId

AND c.coursenumber=e.coursenumber

AND LOWER(c.topic) LIKE '%introduction%';
```

To show the courses I have created I have shown a screenshot below:

```
ij(CONNECTION1)> select * from course;
COURSENUMB&|TOPIC
--------------------------------
1          |University Magic
2          |Introduction Magic
3          |Introduction
4          |Introduction Math

4 rows selected
```

Below is showing the students enrolled in Introduction courses:

```
ij(CONNECTION1)> SELECT * FROM student s, Enrollment e, Course c WHERE s.Student
Id=e.StudentId AND c.coursenumber=e.coursenumber AND LOWER(c.topic) LIKE '%intro
duction%';
STUDENTID  |STUDENTNAME         |STUDENTSTREETADDRESS          |SUBURB
             |COURSENUMB&|STUDENTID  |COURSENUMB&|TOPIC
-------------------------------------------------------------------------------
-------------------------------------------------------------------------
2          |Orange              |22 Smith Steet                |Griffith
             |2          |2          |2          |Introduction Magic
3          |Penny               |11 ith Steet                  |Griffith
             |3          |3          |3          |Introduction
4          |Penny               |11 ith Steet                  |Nathan
             |4          |4          |4          |Introduction Math
```

2.(1 mark) Write (and draw as a tree) the plan for this query using relational algebra operators.

Drawing the query plan for this query.

SELECT studentname

FROM student s, Enrollment e, Course c

WHERE s.StudentId=e.StudentId

AND c.courseNumer = e.couseNumber

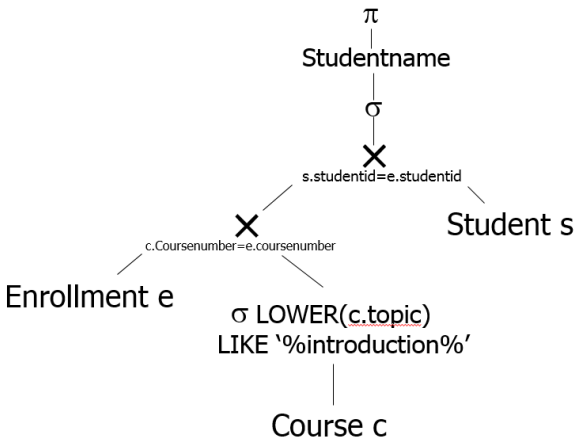
AND LOWER(c.topic) LIKE '%introduction%';

In the logical query plan I have used symbols join and select and projection shown below.

X = join

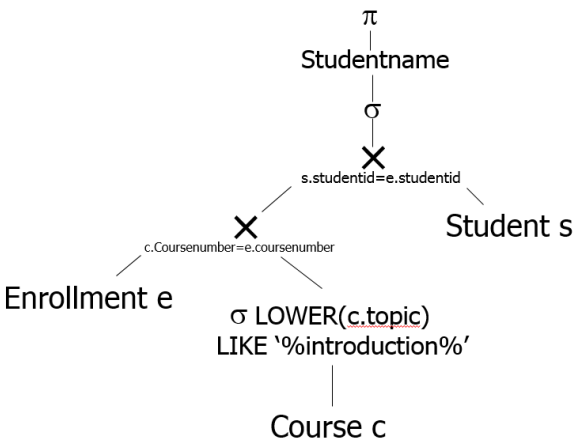
σ = select

π = projection

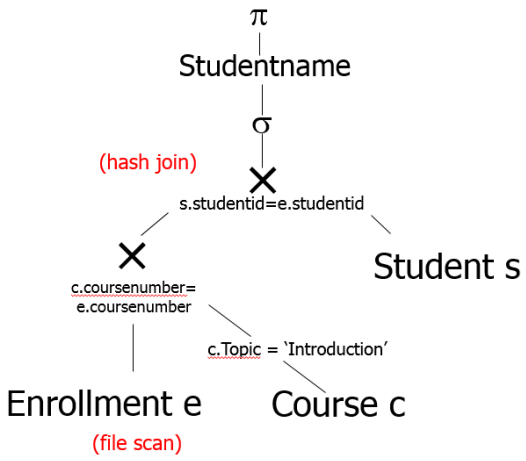


3.(2 marks) Assume that there is a B+ index on each primary key. Write an alternative plan and justify that the alternative plan is faster.

A)



B)



The following reasons why plan B could be faster than plan A:

• The low selectivity of the selection predicate LIKE on A.
B is an equality query which contains three point queries, using primary keys on all but one statement being c.topic.
Hash indexes far exceed b+ trees in relation to point queries for speed making B faster than A.

Hash tables are particularly efficient when the maximum number of entries can be predicted in advance, so that the bucket array can be allocated once with the optimum size and never resized.

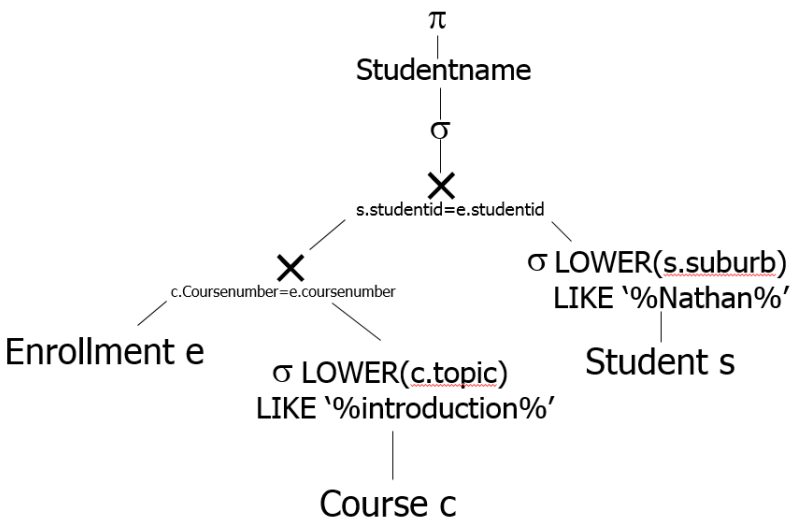
If the set of key-value pairs is fixed and known ahead of time, one may reduce the average lookup cost by a careful choice of the hash function, bucket table size, and internal data structures. , this case the point queries know they are looking for two of the same id's and 'introduction' making a clear direction of where it needs to look.
To further improve this c.topic is still being used as a search key unlike in A.
In C.topic is only looking for results where Introduction is found which requires less effort than using LIKE and wildcard.
• If there are lots of matches, hash joins may be faster than index nested loops, as the hash indexes read its input only once (unless the input is too large). The index-nested loop may end-up reading the same pages of B multiple times.
• Pushing the selection down can get rid of lots of B tuples before the join, reducing the cost of that operation.
However it is important to note b+ trees are generally easier to maintain.

4.(1 marks) What would be two plans for the query "Retrieve the names of students who are enrolled in at least one course with topics starting with INTRODUCTION and live in NATHAN"

The select query performs joins to enable the ability to search tuples which are in different tables. It selects on student name from student table, two like clauses to find introduction and Nathan.

SELECT studentname FROM student s, Enrollment e, Course c
WHERE s.StudentId=e.StudentId
AND c.coursenumber=e.coursenumber
AND LOWER(c.topic) LIKE '%introduction%'
AND LOWER(s.suburb) LIKE '%nathan%';

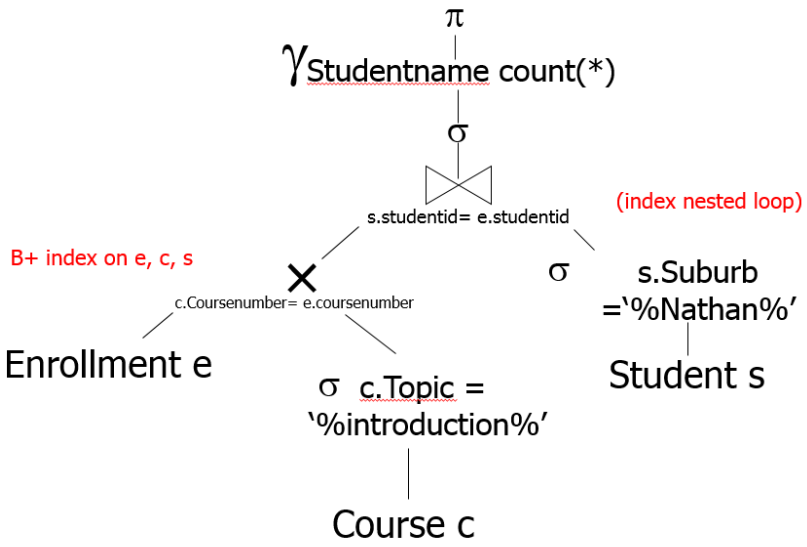
```
ij(CONNECTION1)> SELECT studentname FROM student s, Enrollment e, Course c WHERE
 s.StudentId=e.StudentId AND c.coursenumber=e.coursenumber AND LOWER(c.topic) LI
KE '%introduction%' AND LOWER(s.suburb) LIKE '%nathan%';
STUDENTNAME
--------------------
Penny

1 row selected
```



Query 2 and Plan

```
SELECT S.StudentName FROM Student AS S, Course C, Enrollment e INNER JOIN
Enrollment AS E ON S.StudentId = E.StudentId
 WHERE s.studentId = e.studentId AND s.Suburb = 'Nathan' AND c.Topic = 'Introduction'
GROUP BY S.StudentId, S.StudentName HAVING COUNT(*) = (SELECT COUNT (*)
FROM courses);
```

To improve the original logical query plan, replacing a product followed by a selection with a join. Join algorithms are usually faster than doing product followed by selection on large result of the product.



QUESTION 30: CONSIDER THE NOTION OF *TRANSACTION*.

1. (1 mark) What is a transaction in database management system?

A transaction is an execution of a user program, and is seen by the DBMS as a series or list of actions. The actions that can be executed by a transaction include reads and writes of database objects, whereas actions in an ordinary program could involve user input, access to network devices, user interface drawing, etc.

Using a DBMS to manage data has many advantages:
Data independence: Application programs should be as independent as possible from details of data representation and storage. The DBMS can provide an abstract view of the data to insulate application code from such details.

Efficient data access: A DBMS utilizes a variety of sophisticated techniques to store and retrieve data efficiently. This feature is especially important if the data is stored on external storage devices.
Data integrity and security: If data is always accessed through the DBMS, the DBMS can enforce integrity constraints on the data. For example, before inserting salary information for an employee, the DBMS can check that the department budget is not exceeded. Also, the DBMS can enforce access controls that govern what data is visible to different classes of users.
Data administration: When several users share the data, centralizing the administration of data can offer significant improvements. Experienced professionals who understand the nature of the data being managed, and how different groups of users use it, can be responsible for organizing the data representation to minimize redundancy and for fine-tuning the storage of the data to make retrieval efficient.

2. (1 mark) Are transactions necessary if data is not being shared (justify your answer)? Are transactions needed if all clients are interested in consulting the data, but not modifying (justify your answer)?

Transactions are necessary even if data is not being shared because concurrent execution of user programs is essential for good DBMS performance. This is because disk accesses are frequent, and relatively slow without transactions, it is important to keep the cpu humming by working on several user programs concurrently. A user's program may carry out many operations on the data retrieved from the database, but the DBMS is only concerned about what data is read/written from/to the database. A transaction is the DBMS's abstract view of a user program: a sequence of reads and writes.

A transaction might commit after completing all its actions, or it could abort (or be aborted by the DBMS) after executing some actions. A very important property guaranteed by the DBMS for all transactions is that they are atomic. That is, a user can think of a Xact as always executing all its actions in one step, or not executing any actions at all. DBMS logs all actions so that it can undo the actions of aborted transactions.

Each Xact must obtain a shared lock on object before reading, and an exclusive lock on object before writing. All locks held by a transaction are released when the transaction completes • (Non-strict) 2PL Variant: Release locks anytime, but cannot acquire locks after releasing any lock. If an Xact holds an X lock on an object, no other Xact can get a lock on that object. Strict 2PL allows only serializable schedules. Additionally, it simplifies transaction aborts Non-strict 2PL also allows only serializable schedules, but involves more complex abort processing

If a transaction is aborted, all its actions have to be undone. Not only that, if reads an object last written by must be aborted. Most systems avoid such cascading aborts by releasing a transaction's locks only at commit time. If writes an object, can read this only after commits. In order to undo the actions of an aborted transaction, the DBMS maintains a log in which every write is recorded. This mechanism is also used to recover from system crashes: all active Xacts at the time of the crash are aborted when the system comes back up.

If crash does occur the following the Aries Recovery Algorithm should be performed.

Analysis: Scan the log forward (from the most recent checkpoint) to identify all Xacts that were active, and all dirty pages in the buffer pool at the time of the crash.

Redo: Redoes all updates to dirty pages in the buffer pool, as needed, to ensure that all logged updates are in fact carried out and written to disk. Undo: The writes of all Xacts that were active at the crash are undone (by restoring the before value of the update, which is in the log record for the update), working backwards in the log. (Some care must be taken to handle the case of a crash occurring during the recovery process).

3. (1 mark) What elements of concurrency demand the use of transactions?

A DBMS supports the notion of a transaction, which is conceptually a single user's sequential program. Users can write transactions as if their programs were running in isolation against the database. The DBMS executes the actions of transactions in an interleaved fashion to obtain good performance, but schedules them in such a way as to ensure that conflicting operations are not permitted to proceed concurrently.

Users submit transactions, and can think of each transaction as executing by itself. Concurrency is achieved by the DBMS, which interleaves actions (reads/writes of DB objects) of various transactions. Each transaction must leave the database in a consistent state if the DB is consistent when the transaction begins. DBMS will enforce some ICs, depending on the ICs declared in CREATE TABLE statements. Beyond this, the DBMS does not really understand the semantics of the data. (e.g., it does not understand how the interest on a bank account is computed).

4. (2 marks) Describe what the ACID properties of transactions mean.

Each term is described below.

All or Nothing

(a)Atomicity means a transaction executes when all actions of the transaction are completed fully, or none are. This means there are no partial transactions (such as when half the actions complete and the other half do not).

(b) Consistency involves beginning a transaction with a 'consistent' database, and finishing with a 'consistent' database. For example, in a bank database, money should never be created or deleted without an appropriate deposit or withdrawal. Every transaction should see a consistent database.

Cannot see other transactions

(c) Isolation ensures that a transaction can run independently, without considering any side effects that other concurrently running transactions might have.

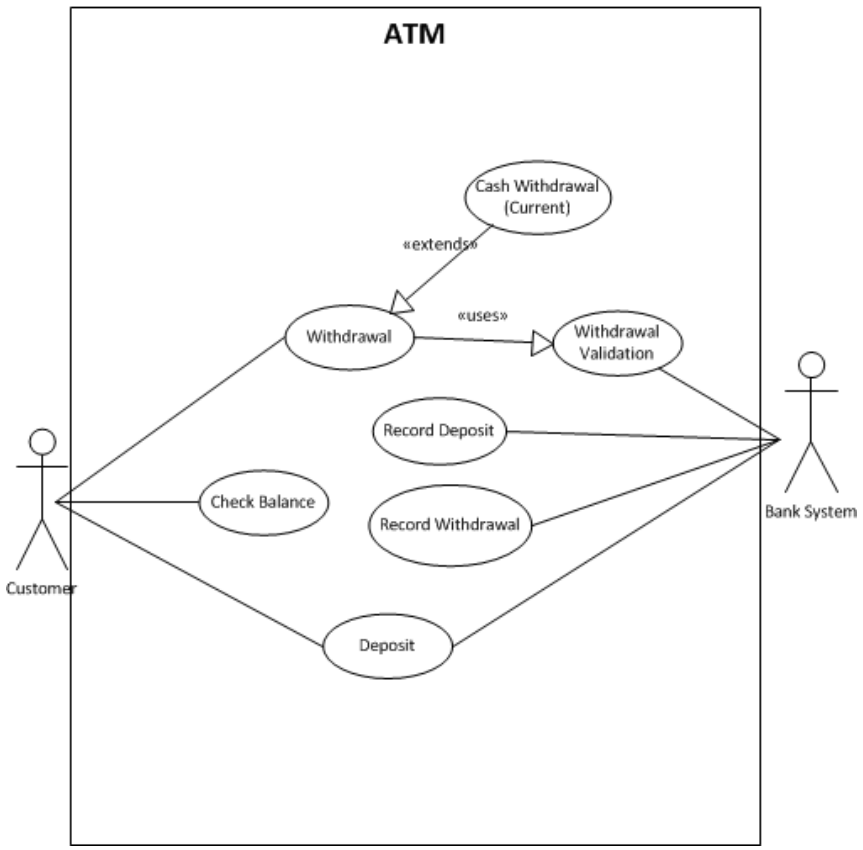
When a database interleaves transaction actions for performance reasons, the database protects each transaction from the effects of other transactions.

Constant State

(d) Durability defines the persistence of committed data: once a transaction commits, the data should persist in the database even if the system crashes before the data is written to non-volatile storage.

(e) A schedule is a series of (possibly overlapping) transactions.

5. (1 mark) Describe a scenario where non-trivial transactions will be needed (there is more than one operation into the database to complete a system function). Draw the corresponding use-case.



A customer withdrawing money from a Bank System

The customer wants to withdraw money from the bank, the current withdrawal amount is sent to the Bank System and compared against the database the Withdrawal validation then checks against the database and it accepts or denies the withdrawal validation, this is then passed back to the customer. If the request is validated the money is withdrawn otherwise the request is denied.

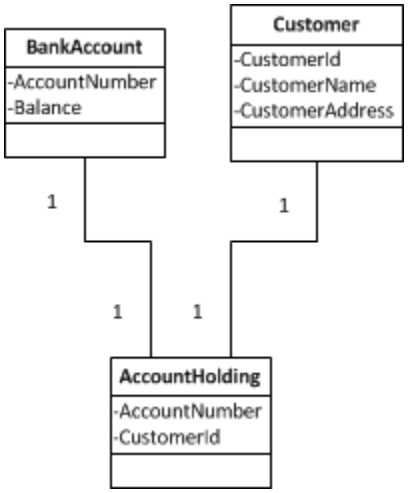
6. (2 marks) Consider the schema

BankAccount (AccountNumber, Balance)

Customer (CustomerId, CustomerName, CustomerAddress)

AccountHolding(AccountNumber, CustomerId)

Provide a small ER diagram of the data model.



7. (4 marks) Provide in Java and JDBC to describe the structure of a client that is used to add a new account to a given customer (with s starting given balance). We must check the customer exists. The customer id, name and address are also given. It is not important that you are complete accurate about java syntax, but it is important you show the structure of the business logic and the handling of exception and all elements that construct and handle the transaction from the client side.

Customers.java is attached to this documentation, for the answer to this problem please refer to this. The create statements have been commented in the java file.

QUESTION 31: CONSIDER FIGURE 2.

1.(4 mark) Write the SQL statement(s) that will find the name (EMP FNAME, EMP LNAME) of the employer and the project (PROJ NAME)for the employer that has most assigned hours to the project, per project.

SELECT emp_lname, emp_fname, proj_name FROM Project p,
Employee e, Assignment a
WHERE a.Proj_num=p.Proj_num
AND e.emp_num = p.emp_num
AND a.assign_hours = (select MAX(assign_hours)
FROM assignment);

```
ij> SELECT emp_lname, emp_fname, proj_name FROM Project p, Employee e, Assignmen
t a WHERE a.Proj_num=p.Proj_num AND e.emp_num = p.emp_num AND a.assign_hours = (
select MAX(assign_hours) FROM assignment);
EMP_LNAME                     |EMP_FNAME                     |PROJ_NAME

------------------------------------------------------------------------------
------------
Tina                          |Turner                        |Updates

1 row selected
```

SELECT emp_lname, emp_fname, proj_name
FROM Project p, Employee e, Assignment a
WHERE p.proj_num IN (
SELECT a.proj_num
FROM a
WHERE a.assign_hours = (select MAX(assign_hours)
FROM a));

2.(1 mark) Write an SQL query that lists all the projects where employee with Employee Number 5723 is a manager.

SELECT PROJ_NUM FROM Project p, Employee e
WHERE p.emp_num = e.emp_num
AND e.emp_num = 5723;

Below shows the select query working.

```
ij> SELECT PROJ_NUM FROM Project p, Employee e WHERE p.emp_num = e.emp_num AND e
.emp_num = 5723;
PROJ_NUM
-----------
3

1 row selected
```

Below is a display of the projects in the database and the employee number of 5723 that is associated with that given project.

```
ij> select * from project;
PROJ_NAME                     |PROJ_NUM    |EMP_NUM
------------------------------------------------------------
amx                           |1           |1
Updates                       |2           |2
anz                           |3           |5723

3 rows selected
ij>
```

3. (2 marks) Describe what are the challenges of having a loop (cycle) by the relationships

- **Employee!(managesa)Project.**
- **Project!(isrequiredbyan)Assignment.**
- **Assignment!(enteredby)Employee**

Challenges to overcome in loop situations

Generally the most common issue with loops is consistency of redundant information being passed from relationships which are not properly handled. Another issue involved with not properly implemented loops is the issue of foreign keys not being correctly used. An example of a Foreign Key in the ER-diagram not being used correctly would be the Foreign Key in employee referenced to Job. Having this foreign key in employee leads to many issues, let alone this occurring in a loop situation would be widely damaging to the database in that type of scenario.

The loop shown above is a dependant loop, a dependant loop contains attributes of interlinking tables in them for the database to function. In this case it is necessary for these attributes to be in the corresponding tables. It may sound like values could end up not NULL values and leave the system open to anomalies, however in this case the loop helps constrain and ensure data redundancy does not occur. In a lost of cases loops are not a good idea and can occur in serious loopholes.

In this ER-Diagram problems of Domain constraints such as not NULL values on Primary Key's and foreign keys are very unlikely as the dependent loop handles this. Referential Keys within corresponding loop tables helps manage and ensure the loop is functioning correctly. A constrained relationship can help maintain other constrained relationships

since each constraint is checked independently using underlying relationship formalizations.

An example of this would be:

- To add a Project but had no Manager you wouldn't be able to do so, because you need to have an employee number to be able to create a project.
- To create an Assignment, project would have to exist as an assignment cannot exist without a project.
- The Assignment must have at least one employee number referenced from Employee for Assignment to exist.

Below I created an Employee number that does not exist the Project has checked against the Employee table to see if this employee number (8) exists. As it did a roll back was issued. Ensuring that values entered are corresponding and data redundancy does not occur.

```
ij> insert into project values('VPN', 5, 8);
ERROR 23503: INSERT on table 'PROJECT' caused a violation of foreign key constra
int 'SQL150711231639121' for key (8).  The statement has been rolled back.
```

4. Use this example to describe one domain constraint, one referential integrity constraint and one key constraint.

A domain Constraint is defined as set of all unique values permitted for an attribute.
An example of this could be Emp_hiredate being set to DATE allowing only insert values of a date being able to be added to the database.
Referential integrity Constraint can reference a column or columns from another table which is referenced by it's primary key. An example of an Referential integrity Constraint would be Project EMP_NUM referencing Employee EMP_NUM.
The one key constraint or Primary Key constraint uniquely identifies each record in a database table. Primary keys must contain unique values and cannot contain NULL Values.

Most tables should have a primary key, and each table can have only one primary key. Primary Key example could be in Project PROJ_NUM.

5. Write the CREATE statement to formulate at least one table with the 3 types of constraints indicated above. Explain the benefits of integrity constraints.

Below is the create statement I used to Create the Project Table and its attributes. It contains a Primary key known as proj_num and references another Primary key from employee (emp_num).

CREATE TABLE PROJECT(PROJ_NUM INT, PROJ_NAME VARCHAR (20), EMP_NUM INT
PRIMARY KEY(PROJ_NUM), FOREIGN KEY(EMP_NUM) REFERENCES EMPOYEE(EMP_NUM));

- One key Constraint – setting proj_num to Primary Key
- Domain Constraint – setting proj_num to int

- Referential Integrity Constraint – referencing employee emp_num for project emp_num

Integrity constraints ensure data integrity, this is of high importance when using DBMS. If used correctly it ensures data anomalies do not occur and ensure data is updated and remains in a constant state from table to table.

Integrity constraints are simple to create and maintenance is always enforced, regardless of tool or application that updates table data and performs faster than other methods. The importance of integrity constraints in DBMS is paramount.
In this case specifically the importance of the foreign key referencing employee ensures the manager exists and is corresponding to the employee table. Having a Primary key for the Proj_num ensures each project is unique and having corresponding input fields such as int and varchar and size lengths ensure users will put in the correct values to the database.

6. (1 mark) Write the SQL statement that finds the name of each employee (from Employee) and their job description (from Job).

SELECT EMP_LNAME, EMP_FNAME JOB_DESCRIPTION FROM EMPLOYEE e, JOB j WHERE e.JOB_CODE = j.JOB_CODE;

```
ij> SELECT EMP_LNAME, EMP_FNAME JOB_DESCRIPTION FROM EMPLOYEE e, JOB j WHERE
 e.JOB_CODE = j.JOB_CODE;
EMP_LNAME                     |JOB_DESCRIPTION
------------------------------------------------------------------------------
Rowmn                         |Mandy

1 row selected
```

7. (3 marks) Describe 3 possible (alternative) algorithms that a database may use to compute the answer in the previous paragraph.

Some algorithms that a database may use to compute the answer in the previous paragraph could be:

Hash tables, linked lists, page abstraction - merge sort

The simplest file structure is an unordered file or heap file.

One possibility is to maintain a heap file as a doubly linked list of pages. The DBMS can remember where the first page is located by maintaining a table containing pairs of ⟨heap file name, page 1 address in a known location on disk. We call the first page of the file the header page.

Clustered Indexes

When a file is organized so that the ordering of data records is the same as or close to the ordering of data entries in some index, we say that the index is clustered; otherwise, it clustered is an unclustered index.

The cost of using an index to answer a range search query can vary tremendously based on whether the index is clustered. If the index is clustered, i.e.,we are using the search key of a clustered file, the rids in qualifying data entries point to a contiguous collection of records, and we need to retrieve only a few data pages. If the index is unclustered, each qualifying data entry could contain a rid that points to a distinct data page, leading to as many data page l/Os 8.'3 the number of data entries that match the range selection.

Hash-Based Indexing

We can organize records using a technique called hashing to quickly find records that have a given search key value. For example, if the file of employee records is hashed on the name field, we can retrieve all records about Mary. Hash Based is good for equality selections.

In this approach, the records in a file are grouped in buckets, where a bucket consists of a primary page and, possibly, additional pages linked in a chain.
The bucket to which a record belongs can be determined by applying a special function, called a hash function, to the search key. Given a bucket number, a hash-based index structure allows us to retrieve the primary page for the bucket in one.

Tree Based Indexing

An alternative to hash-based indexing is to organize records using a treelike data structure. The data entries are arranged in sorted order by search key value, and a hierarchical search data structure is maintained that directs searches to the correct page of data entries. The lowest level of the tree, called the leaf level, contains the data entries.
This structure allows us to efficiently locate all data entries with search key values in a desired range. All searches begin at the topmost node, called the root, and the contents of pages in non-leaf levels direct searches to the correct leaf page. Non-leaf pages contain node pointers separated by search key values.

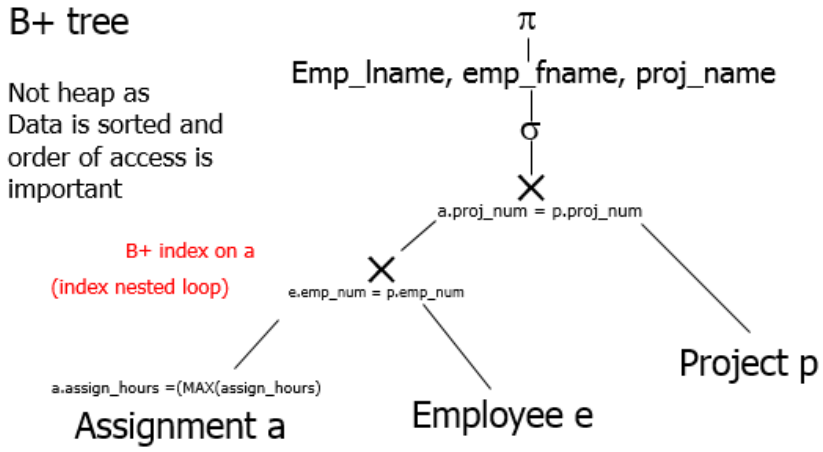
The node pointer to the left of a key value k points to a subtree that contains only data entries less than k. The node pointer to the right of a key value k. Finding the correct leaf page is faster. Than binary search of the pages in a sorted file because each non~leaf node can accommodate a very large number of node-pointers, and the height of the tree is rarely more than three or four in practice.Points to a subtree that contains only data entries greater than or equal to k.

Notes:

- **Selections usually reduce the size of the relation**
- **Usually good to do selections early, i.e., "push them down the tree"**

An example of a b+ index

SELECT emp_lname, emp_fname, proj_name
FROM Project p, Employee e, Assignment a
WHERE p.proj_num IN
(SELECT a.proj_num FROM a
WHERE a.assign_hours = (MAX(assign_hours) FROM a));



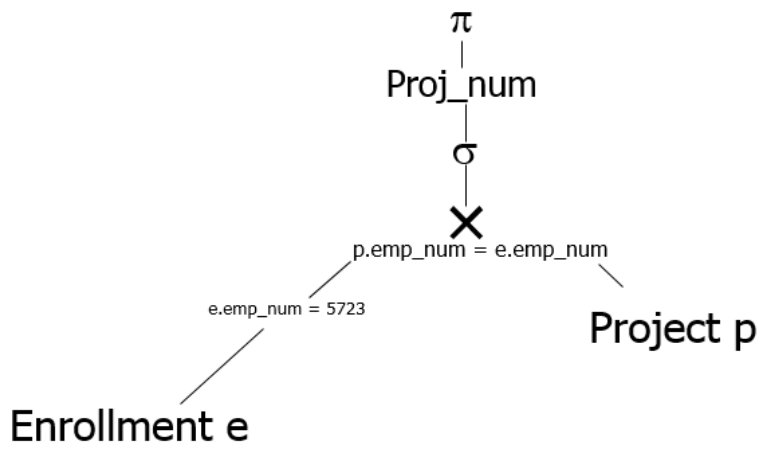
In this way, by walking down the nodes, doing comparisons along the way we can avoid scanning thousands of records, in just a few easy node scans. Hopefully this diagram helps to illustrate the idea. This is qualified as a range query as some values are not constant unlike an equality query.

Heap Files

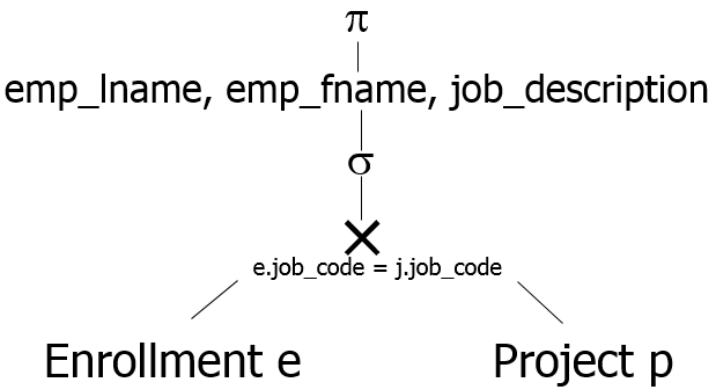
SELECT PROJ_NUM FROM Employee e, Project p
WHERE e.emp_num = 5723; -- point query
AND p.emp_num = e.emp_num;

For a query to be used with association of Heap Files it must be able to meet Equality selection and have exactly one match. Equality involves comparing two values which, in this query it occurs twice. There will only be one emp_num with the number 5723 and one p.emp_num = e.emp_num. Making this query usable with Heap Files.
Heap Files Insert always at end of file.

The use emp_num comparison definitely improves query performance by searching by a constant value ‘5723’.



SELECT EMP_LNAME, EMP_FNAME JOB_DESCRIPTION
FROM EMPLOYEE e, JOB j
WHERE e.JOB_CODE = j.JOB_CODE;



Further Indexing Information

Indexes with Composite Search Keys

Equality query: Every field value is equal to a constant value. E.g. index:
emp_lname='Smith' AND emp_fname ='Mary';

Range query: Some field value is not a constant. E.g.: emp_lname ='Smith'; or
emp_lname ='Smith'and emp_num > 1

H -Time required to apply the hash function to a record

C- Compare a field value to a selection constant

R - To denote the number of records per page

D - Average time to read or write a disk page

B - To denote the number of data pages when records are packed onto pages with no wasted space

F - To denote the fan-out on tree indexes, which typically is at least 100

Heap Files

Equality selection on key; exactly one match.
Search with Equality Selection: Suppose that we know in advance that exactly one record matches the desired equality selection, that is, the selection is specified on a candidate key. On average, we must scan half the file, assuming that the record exists and the distribution of values in the search field is uniform.

For each retrieved data page, we must check all records on the page to see if it is the desired record. The cost is O.5B(D + RC). If no record satisfies the selection, however, we must scan the entire file to verify this. If the selection is not on a candidate key field (e.g., "Find emp_num = 5723"), we always have to scan the entire file because records with emp_num = 5723 could be dispersed all over the file, and we have no idea how many such records exist.

Search with Range Selection: The entire file must be scanned because qualifying records could appear anywhere in the file, and we do not know how many qualifying records exist. The cost is B(D + RC).

Sorted Files

Files compacted after deletions.
Scan: The cost is B(D +RC) because all pages must be examined. Note that this case is no better or worse than the case of unordered files. However, the order in which records are retrieved corresponds to the sort order, that is, all records in order.

Search with Equality Selection: We assume that the equality selection matches the sort order. In other words, we assume that a selection condition is specified on the first field in the composite key (e.g., emp_num = 1527). If not (e.g., emp_lname = ‘Ranar’), the sort order does not help us and the cost is identical to that for a heap file.

8. (3 marks) Describe how indexes can be used to speed up the queries for 31-1, 31-2, and 31-6.

Depending on the indexing you want to perform and the data you are wanting to retrieve query selection could be better or worse. For Question 1,2 and 6 indexes have been used to further optimize accessing corresponding table data. By using indexes saves the need to use multiple select statements and use unions which can be very time costly on the database.
When using indexes it is important to evaluate plans with the lowest cost. An important thing to note is to ensure sparsity of indexes instead of index density. The reason for this is by having sparsity of data less data will have to be searched, meaning quicker queries. When using b+ index files data is split and split again depending how big the b+ tree is making further index sparsity.
By using multiple level indexes the sparsity of data is spread making query optimization quicker and easier to sort through. With b+ trees the data is also automatically sorted making further query optimization.

When creating the following selections these factors where considered.

Selections:

- Push down tree as far as possible
- If condition is an AND, split and push separately
- Sometimes need to push up before pushing down

Projections:

- Can be pushed down
- New ones can be added (but be careful)

Duplicate elimination:

Sometimes can be removed

- Selection/product combinations:
- can sometimes be replaced with join

Query Examples

SELECT emp_lname, emp_fname, proj_name
FROM Project p, Employee e, Assignment a
WHERE p.proj_num IN (SELECT a.proj_num FROM a
WHERE a.assign_hours = (select MAX(assign_hours) FROM a));

By using p.proj_num this indexing on project makes it quicker to find the max assigned hours. Although b+ indexing is quick, using the max ensures having to sort through all the corresponding records for the largest of each project which is time costly.

However because we are only comparing project, assign hours size against only those employees in that project not the whole table makes this is a lot quicker instead searching all of the rows by size and then finding the corresponding project_num which would be far more time costly.

Some benefits of using b+ tree index files are it automatically reorganizes itself with small, local, changes, in the face of insertions and deletions. Reorganization of entire file is not required to maintain performance.

SELECT PROJ_NUM FROM Project p, Employee e
WHERE p.emp_num = e.emp_num
AND e.emp_num = 5723;

By using an equality query we can check if emp_num = 5723 enabling the ability to find all primary keys with 5723 for the id. As this is on a primary key it makes searching quite fast.

To further optimize this query however this query could be broken down into one where statement p.emp_num = 5723, as emp_num will already exist as an employee there should not be any need to check if p.emp=num e.emp_num, as p.emp_num is a foreign key which references employee. This could vary slightly in speed as p.emp_num is a secondary indices unlike e.emp_num which is primary hence why the statement has been kept the same.

SELECT EMP_LNAME, EMP_FNAME JOB_DESCRIPTION
FROM EMPLOYEE e, JOB j
WHERE e.JOB_CODE = j.JOB_CODE;

For this query it is necessary to join both of the tables as we pulling the emp_lname, emp_fname and job_description. By using indexes we can easily check if the e.job_code is the same the j.job_code instead of having to check both tables by using multiple select statements with a union which can be incredibly time costly. The use of indexes speeds up accessing tables a lot when used correctly and saves the need to add multiple select statements.

Further Query Algorithms For Q1,2 & 6

For **Question 1** because ID's are on every table it would be quicker using Sorted Files instead of searching with heap files as there maybe more than one row, e.g. There could be two employees with the same amount

of hours and we are searching by assign_hours which is not a primary key.
Which could also make this method the same speed as heap sorting.

For **Question 2** projects managed by emp_num = 5723, if there is just one employee with the ID 5723 Heap Sort could be the best choice, if there is exactly one match, this would be the fastest choice. If there was a duplicate value primary key issues could occur.

In the case of more than one value is present Search with Equality Selection would be preferred. We would assume that the equality selection matches the sort order. In other words, we assume that a selection condition is specified on the first field in the composite key which would be the emp_num from employee and emp_num from project.

For **Question 6** as it only looking for the emp_num and job code which are both primary keys and there is only one job for one employee a heap file could be fast because there should be only one match for this given query. However because Job is not linked correctly issues may be incurred.

9. (4 marks) Because every EMPLOYEE must have a JOB (with a job description), describe schematically what are the READ and WRITE operations (that shall be held as a transaction) when an application is adding a new employee.

The file to demonstrate this has been attached called Employee.java to display this answer in more detail, view the file. The file contains the associated create and insert statements.

T1: BEGIN Job= Job_code=1, Job_description=newjob, job_chg_hour+300 END
T2: BEGIN Employee= Emp_lname=Jane, Emp_fname=Smith, Emp_inital=JM, Emp_HireDate=04/03/2015, Job =Job_code END

T1: R(Job), W(Job), R(Employee), W(Employee),
T2: R(Job), W(Job), R(Employee), W(Employee),

There is no guarantee that T1 will execute before T2 or vice-versa, if both are submitted together. However, the net effect must be equivalent to these two transactions running serially in some order.
The problem is that T1 works on the basis of employee not being about to exist without Job as Employee references Job. When employee is created the foreign key would not exist or Job must already exist without a corresponding Employee which can leave to huge data redundancy if Job is not matched to employee. So this interleaving cannot be equivalent to first reading and writing employee and then reading and writing to disk job. As this would not effectively work with this system.

After doing this it is a lot easier to create job first and match it to employee after. This is not a good demonstration of good system design as it makes it hard to add the Foreign Key into Employee unless Job already exists. To resolve this you would have a foreign key in Job of Emp_num and drop the column job_code from employee.

To assist this Deadlock prevention can help ensure deadlocks are less likely this could be done by assigning priorities based on timestamps. Assume Ti wants a lock that Tj holds. Two policies are possible: Wait-Die: It Ti has higher priority, Ti waits for Tj; otherwise Ti aborts Wound-wait: If Ti has higher priority, Tj aborts; otherwise Ti waits If a transaction re-starts, make sure it has its original timestamp.

QUESTION 32: CONSIDER THE UML DIAGRAM IN FIGURE 4.

1. **(3 marks) Provide the resulting relational schema when the data elements in the IS A relationship of the classes of Figure 4 are mapped into a relational data model using each of the 3 methods to perform such a transformation. Note: The method teach() retrieves the students that are taught by a professor. Also, a student can be taught by one or more professors.**

All create statements and queires have been tested and work as expected in derby.

NOTE: The following Assumptions have been made based off Fig 4. UML Class diagram.

- Person is not modelled as an Abstract Superclass, as the Class Name is not in Italics.
- Person, Student and Professor are all Concrete Classes.
- Student and Professor can be considered as Concrete Leaf Classes.
- Student and Professor are sub-classes of Person. (Inheritance)
- Student has a 1 to Many relationship with Professor, this implies a constraint where a Student must have at least 1 Professor.
- Professor has a 1 to Many relationship with Student, this implies a constraint where a Professor must have at least 1 Student.
- Due to the stated nature of the teach() method and the 1..M M..1 relationship between Student and Professor, it can be assumed that there will be a juntion/mapping table that contains the students id and the professor's id. This will allow the teach() method to query the junction table to retrieve the set of student's that have been mapped to the Professor's id.

- get_gpa() implies that a GPA attribute exists in the Student Table, but it is not shown in the attribute field in the Student Class.
- get_gpa() method is assumed to refer to the students cumulative gpa and not to a specific course gpa as there is no Course Class in the provided UML Class Diagram and no reference to a Course Table existing.

Method 1)

- Class Table Inheritence: One Table for each Concrete Class

Note: - Person Table is given a unique id called person_id, which Student and Professor Tables will inherit.
- Student and Professor Tables have FK's references the Person Table.
- Junction/Mapping Table is created for the 1..M M..1 relationship between Professor and Student.
- GPA Coloumn is added to the Student table.
- id in Student Class Diagram has been renamed to s_id in the Student Table;
- This schema allows for a person to be a student and a professor. Due to this schema design the junction table currently allows for a professor to teach himself.

SQL Create Statements:

```
CREATE TABLE PERSON(
        person_id int NOT NULL,
        age int NOT NULL,
        PRIMARY KEY (person_id)
);

CREATE TABLE PROFESSOR(
        person_id int NOT NULL,
        title varchar(40) NOT NULL,
        PRIMARY KEY (person_id),
        FOREIGN KEY(person_id) REFERENCES
PERSON(person_id)
);

CREATE TABLE STUDENT(
        person_id int UNIQUE NOT NULL,
        s_id varchar(10) UNIQUE NOT NULL,
        gpa float NOT NULL,
        PRIMARY KEY (person_id,s_id),
        FOREIGN KEY(person_id) REFERENCES
PERSON(person_id)
);

CREATE TABLE STUDENT_PROFESSOR_MAPPING(
        map_id int NOT NULL,
        s_id varchar(10) NOT NULL,
        s_person_id int NOT NULL,
        professor_id int NOT NULL,
        PRIMARY KEY (map_id),
```

```
        FOREIGN KEY (s_person_id,s_id) REFERENCES
STUDENT(person_id,s_id),
        FOREIGN KEY (professor_id) REFERENCES
PROFESSOR(person_id)
    );

    /* get_gpa(); */

        SELECT gpa
        FROM STUDENT
        WHERE s_id='0123456789';

    /* get_age();
        For PERSON  */

        SELECT age
        FROM PERSON
        WHERE person_id = 1;

    /* get_age();
        For STUDENT */

        SELECT age
        FROM PERSON p, STUDENT s
        WHERE p.person_id = s.person_id
        AND s.s_id = '0123456789';

    /* get_age();
        For PROFESSOR */

        SELECT age
        FROM PERSON p, PROFESSOR pr
        WHERE p.person_id = pr.person_id;
```

Teach(); Only returns one instance of where the professor teaches the student, this has been done as the junction table may contain the student and professor more then once. This has been allowed as it is assumed that a professor may teach a student in more then one class and a student can have multiple professors from other classes that they are enrolled in.

```
SELECT DISTINCT s_id
FROM STUDENT_PROFESSOR_MAPPING
WHERE professor_id = 1;
```

```
/* QUERIES THAT WILL FAIL */
INSERT INTO PERSON VALUES(2,23);
INSERT INTO STUDENT VALUES(2,'0123456789',5.5);
INSERT INTO PROFESSOR VALUES(100,'THIS WILL FAIL');

/* SELECT QUERIES */
SELECT * FROM STUDENT_PROFESSOR_MAPPING;
SELECT * FROM STUDENT;
SELECT * FROM PERSON;
SELECT * FROM PROFESSOR;
```

Method 2) - Concrete Table Inheritance:

A) One Flattened Table for each Concrete Class

Tables Used: PERSON, PROFESSOR, STUDENT, STUDENT_PROFESSOR_MAPPING

Note: - Allows for a Person to exist whilst being a Student or a Professor.

B) One Flattened Table for each Leaf Concrete Class

Tables Used: PROFESSOR, STUDENT, STUDENT_PROFESSOR_MAPPING

- Does not allow a Person to exists without being a Student or a Professor.
- Means the get_age() method for Person cannot be completed.

This table is only created if following 2.A method, follow the instructions in Tables Used.

```
CREATE TABLE PERSON(
        person_id int NOT NULL,
        age int NOT NULL,
        PRIMARY KEY (person_id)
);

CREATE TABLE PROFESSOR(
        person_id int  NOT NULL,
        title varchar(40) NOT NULL,
        age int NOT NULL,
        PRIMARY KEY (person_id),
);
```

```
CREATE TABLE STUDENT(
        person_id int UNIQUE NOT NULL,
        s_id varchar(10) UNIQUE NOT NULL,
        gpa float NOT NULL,
        age int NOT NULL,
        PRIMARY KEY (person_id,s_id)
);

CREATE TABLE STUDENT_PROFESSOR_MAPPING(
        map_id int NOT NULL,
        s_id varchar(10) NOT NULL,
        s_person_id int NOT NULL,
        professor_id int NOT NULL,
        PRIMARY KEY (map_id),
        FOREIGN KEY (s_person_id,s_id) REFERENCES
STUDENT(person_id,s_id),
        FOREIGN KEY (professor_id) REFERENCES
PROFESSOR(person_id)
);

/* get_gpa(); */

        SELECT gpa
        FROM STUDENT
        WHERE s_id='0123456789';

/* get_age();
```

For PERSON

A) POSSIBLE - See below.

B) NOT POSSIBLE - Due to the schema used when implementing Concrete Table Inheritance. Where we have created one flattened table for each leaf Concrete Class, a Person does not exists without being a Student or a Professor.

A) Only works if following 2.a method

```
SELECT age
FROM PERSON
WHERE person_id = 1;

/* get_age();
   For STUDENT */

SELECT age
FROM STUDENT
WHERE s_id = '0123456789';
/* OR using person_id */
SELECT age
FROM STUDENT
WHERE person_id = 1;

/* get_age();
   For PROFESSOR */

SELECT age
FROM PROFESSOR
WHERE person_id = 1;
```

teach(); Only returns one instance of where the professor teaches the student, this has been done as the junction table may contain the student

and professor more then once. This has been allowed as it is assumed that a professor may teach a student in more then one class and a student can have multiple professors from other classes that they are enrolled in.

```
SELECT DISTINCT s_id
FROM STUDENT_PROFESSOR_MAPPING
WHERE professor_id = 1;
```

Method 3) - Single Table Inheritance: One Table for all classes in the UML Diagram.

NOTE - Allows for Person to Exist.

```
CREATE TABLE STUDENT_PROFESSOR(
    person_id int NOT NULL,
    title varchar(40),
    s_id varchar(10),
    gpa float,
    age int NOT NULL,
    PRIMARY KEY (person_id)
);

CREATE TABLE STUDENT_PROFESSOR_MAPPING(
    map_id int NOT NULL,
    s_person_id int NOT NULL,
    professor_id int NOT NULL,
    PRIMARY KEY (map_id),
    FOREIGN KEY (s_person_id) REFERENCES
STUDENT_PROFESSOR(person_id),
    FOREIGN KEY (professor_id) REFERENCES
STUDENT_PROFESSOR(person_id)
);

/* get_gpa(); */
```

```
SELECT gpa
FROM STUDENT_PROFESSOR
WHERE s_id='0123456789';

/* get_age();
    For PERSON
```

1st query is used if you wish to ensure that the person is only a person and not a student or a professor. 2nd query is used if it does not matter if the person is a student or professor.

1)

```
SELECT age
FROM STUDENT_PROFESSOR
WHERE person_id = 3
AND title IS NULL
AND s_id IS NULL
AND gpa IS NULL;
```

2)

```
SELECT age
FROM STUDENT_PROFESSOR
WHERE person_id = 3;
```

```
/* get_age();
    For STUDENT */

    SELECT age
    FROM STUDENT_PROFESSOR
    WHERE s_id = '0123456789';
```

```
/* get_age();
        For PROFESSOR */

        SELECT age
        FROM STUDENT_PROFESSOR
        WHERE person_id = 2
        AND title IS NOT NULL;
```

/* teach(); Only returns one instance of where the professor teaches the student, this has been done as the junction table may contain the student and professor more then once. This has been allowed as it is assumed that a professor may teach a student in more then one class and a student can have multiple professors from other classes that they are enrolled in.

```
        SELECT s_id
        FROM STUDENT_PROFESSOR
        WHERE person_id IN (
                SELECT s_person_id
                FROM STUDENT_PROFESSOR_MAPPING
                WHERE professor_id = 2);

SELECT * FROM STUDENT_PROFESSOR;
SELECT * FROM STUDENT_PROFESSOR_MAPPING;
```

2.Provide one advantage and one disadvantage of each of the mappings

Class Table Inheritance:

Pro: Closest approximation to the Object Oriented Model, it also can support multiple inheritance. Using Fig 4 as example, it allows for a person to be both a student and a professor.

Con: Operations are slower for most queries and updates. Have to query multiple tables for the object's (entities) data as they are divided into different tables.

Concrete Table Inheritance:

Pro: Efficient operations on the subclasses as the data for the subclasses all exist within the same table.

Con: If changes are made to the superclass, we must also modify the schema in all the subclasses. Difficult to manage roles that derive from multiple inheritance. E.g. Using Fig 4. As an example, a student who is also a professor, which would also result in duplicated data shared from the super class e.g. Using Fig 4. As an example – age.

Single Table Inheritance:

Pro: All data in one table allows for fast and easy retrieval. It is also easy to implement.

Con: High coupling and poor space usage. Multiple rows with NULL values.

2. (2 marks) Provide a query whose answer is faster to calculate by one of the resulting schema but slower by the other two.

A query where we search for all people that are both a Student and a Professor will perform the fastest when using the Single Table Inheritance Schema. The reason for this is that the query only needs to search in one table, whereas if we were to attempt this in a Concrete Table Inheritance

or Class Table Inheritance schema we would have to query both the Student Table and the Professor Table. It should be noted that the Single Table Inheritance will provide the fastest access if you were to list all the objects.
The below query is for the Single Table Inheritance Schema:

```
SELECT person_id
FROM STUDENT_PROFESSOR
WHERE title IS NOT NULL
AND s_id IS NOT NULL
AND gpa IS NOT NULL;
```

The below query demonstrates how to retrieve the same answer but slower under a Concrete Table or Class Table Inheritance schema. This is assuming that person_id is unique across Student and Professor.

```
SELECT person_id
FROM STUDENT s, PROFESSOR pr
WHERE s.person_id = pr.person_id;
```

4. (2 marks) Describe the balance between normalization and de-normalization that may result from the different alternatives to carry a mapping of an object-oriented data model with inheritance to a relational data model.

Denormalisation: Allows performance gains with certain queries, as the DBMS can query just one table for the answer and will not have to query multiple tables and create joins if it was normalised. However there is also the possibility of duplicated data occurring, which is why there is a performance increase with certain queries.
Normalisation: Allows for a schema to match the Object Model more closely and model these relationships between tables. There is no duplicated data occurring, however certain queries will be slower as multiple tables will need to be queried and joined to retrieve the answer.

5. (3 marks) Contrast the fact that in a relational database it is somewhat required to represent the age of a person as an integer attribute, while with object-orientation, the age could be (as suggested by the diagram) a method (and in fact a calculation of the age from the date of birth and the current date). Discuss the advantages provided by the object model.

Advantages of Object Model: By storing the D.O.B of a person instead of storing the age, you will never need to update the existing age in the database. If the age is only stored then once a year you would have to update the field to reflect the persons current age. If no D.O.B is stored then you would have difficulty maintaining the correct age of a person as the database would be incrementing the age once a year on a date that might not be their D.O.B. Also performance gains can be gained by using indexes on the D.O.B as this should normally never need to change.

Note: It is possible for a relational database to store the d.o.b and with a sql query return the persons age, based on the current date.

QUESTION 33: CONSIDER THE DECISION TREE IN FIGURE 5 TO PREDICT RAIN TOMORROW BUILT FORM THE DATABASE IN TABLE 6.

1. (2 marks) Classify the data set using the decision tree.

Q1. ID 1 - Yes

ID 2 - No

ID 3 - No

ID 4 - No

ID 5 - Yes

ID 6 - Yes

ID 7 - Yes

ID 8 - Yes

2. What would be the tests sets and training sets if we apply 4-way cross validation.

Cross Validation Strategy: Re-use is the strategy

In most validation testing 80% is training or learning more about the data and 20% of the data is used for testing. For further in-depth understanding of the data we could use sample re-use we divide the data into many 90/10 train/test partitions and repeat the modelling and testing. So for the 8 examples as is it has been specified to split into a 4 way cross validation this will be done 4 times by doing this we will be able to test all the data sets provided. Giving a clearer understanding of the testing and training sets. This would involve rotate through each row of data until four test sets have been performed.

Not just one part of the data segmented these tests are arbitrary. Sample Re-use enables the ability to test more data and gain a clearer understanding of not just the test set results but the training results also.

Say you have input/outputs for: 1, 2, 3, 4, 5, 6, 7, 8

You would do:

1, 2 || 3, 4, 5, 6, 7, 8

3, 4 ||1, 2, 5, 6, 7, 8

5, 6 || 1, 2, 3, 4, 7, 8

7, 8 || 1, 2, 3, 4, 5, 6

3.(2 marks) Represent the tree as decision rules.

ID1.
IF (Outlook = Overcast)
AND Rain Tomorrow = Yes

ID2.
IF (Outlook = Sunny) ^ (Humidity = High)
AND Rain Tomorrow = No

ID3.
IF (Outlook = Rain) ^ (Wind = Strong)
AND Rain Tomorrow = No

ID4.
IF (Outlook = Rain) ^ (Wind = Strong)
AND Rain Tomorrow = No

ID5.
IF (Outlook = Rain) ^ (Wind = Weak)
AND Rain Tomorrow = Yes

ID6.
IF (Outlook = Overcast)
AND Rain Tomorrow = Yes

ID8.
IF (Outlook = Sunny) ^ (Humidity = Normal)
AND Rain Tomorrow = Yes

SAME THING DIFFERENT WAY

ID1.
IF (Outlook = Overcast)
THEN Rain Tomorrow = Yes

ID2.
IF (Outlook = Sunny) ^ (Humidity = High)
THEN Rain Tomorrow = No

ID3.
IF (Outlook = Rain) ^ (Wind = Strong)
THEN Rain Tomorrow = No

ID4.
IF (Outlook = Rain) ^ (Wind = Strong)
THEN Rain Tomorrow = No

ID5.
IF (Outlook = Rain) ^ (Wind = Weak)
THEN Rain Tomorrow = Yes

ID6.
IF (Outlook = Overcast)
THEN Rain Tomorrow = Yes

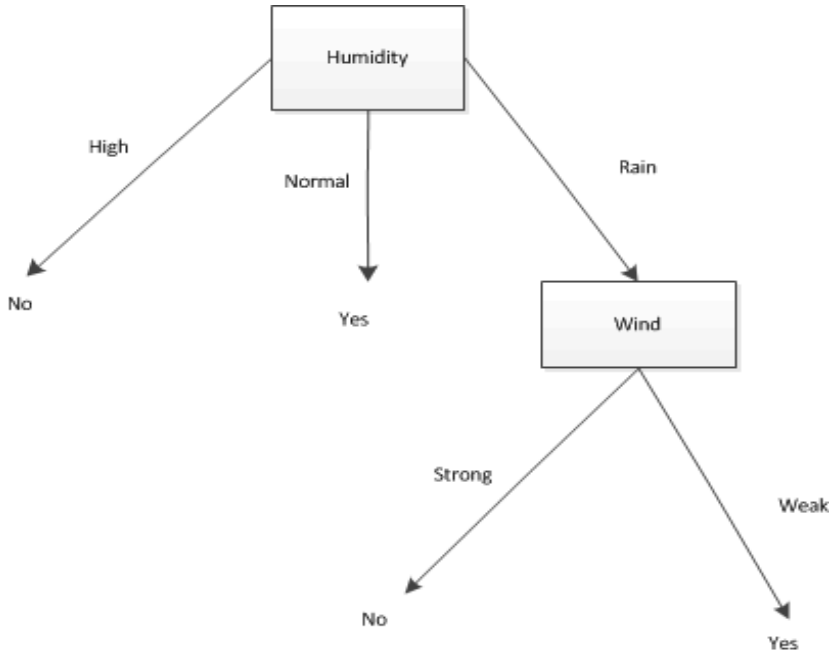
ID8.
IF (Outlook = Sunny) ^ (Humidity = Normal)
THEN Rain Tomorrow = Yes

4. (3 marks) Construct a new decision tree that has HUMIDITY as the root and classifies the data perfectly. Why is not wise to classify the given data perfectly?

It is not wise to classify a decision tree perfectly as it reduces the data for training sets (learning about the data and possible corresponding relationships) and reduces the test data. By making Humidity the Root we have removed possible relationships such as Overcast, which is still corresponding to whether it will rain or not. By removing this we are removing possible data which can give better knowledge if it going to rain tomorrow.

By removing this we have dropped our understanding of the weather forecast and made it more unpredictable. This in turn means less data to test and train which in some cases could be good but in this instance gives us less knowledge.

As 80% is used for training sets and 20% is used for test sets. In this case there are two Overcast rows these would be discarded resulting in removing 20% of the fact table data, which is quite substantial and can make the results more susceptible to being incorrect.


Humidity
High
Normal
Rain
No
Yes
Wind
Strong
Weak
No
Yes