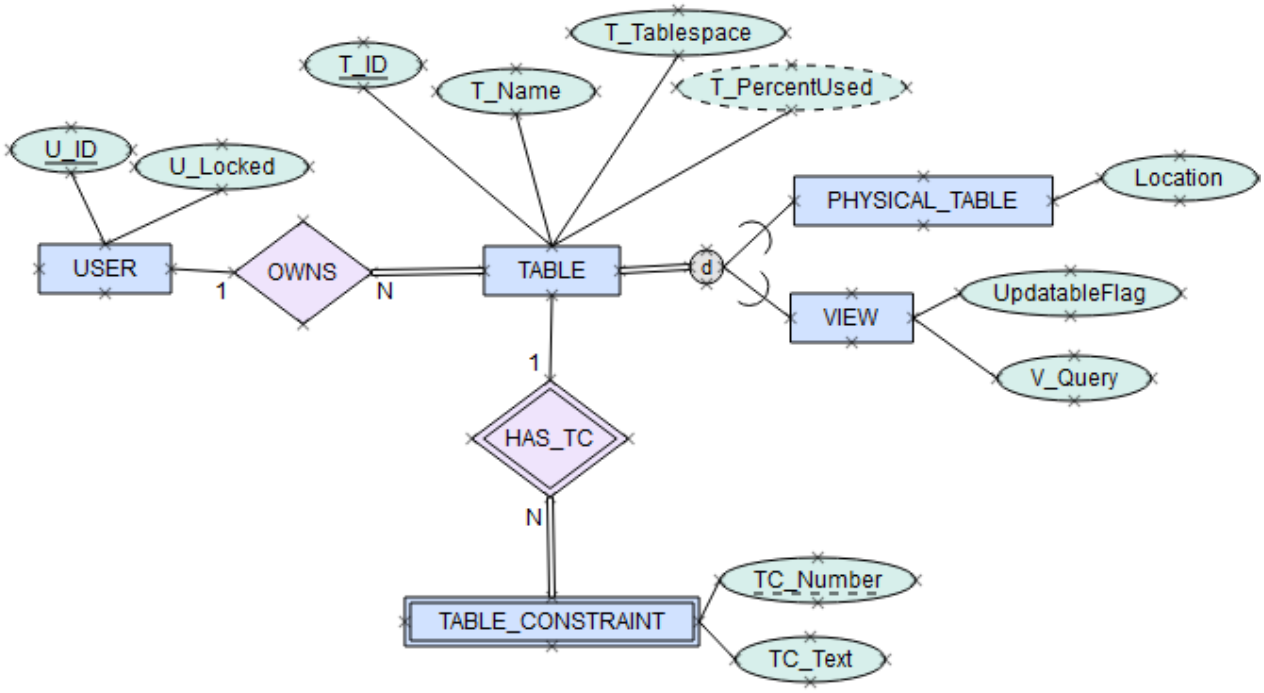
1 [11 marks for the whole question] EER diagrams can model scenarios and concepts at many levels of abstraction. Here we have an EER diagram that models a small part of a relational database management system. This diagram thus represents some of the metadata of that DBMS.

Map this diagram to the equivalent relational database schema. Specify all attributes and primary keys, and note any foreign keys for each relation as well.



Answer:
Table: T_ID, T_Name, T_Tablespace, T_PercentUsed, User (FK)

Physical_Table: T_ID, Location

View: T_ID, Updatable_Flag, V_Query

User: U_ID, U_Locked

Table_Constraint: T_ID, TC_Number, TC_Text

2 [17 marks for the whole question] Relation R (A, B, C, D, E, F) is given, as well as the minimal set of functional dependencies F = {B → C, B → F, D → A, D → E, E → A}.

a) [5 marks] Identify all candidate keys for relation R. Justify your answer.

Answer: Any candidate key must contain B and D, as they're not on any RHS

Test BD: BD+ => BDCFAE ; is CK

So BD is only candidate key, as this is only two-letter combination with BD

(could also do by testing all single letter, two letter, and three letter closures)

b) [5 marks] Identify the highest normal form that R satisfies. Justify your answer.

Answer: We'll assume R is in 1NF by assuming none of the attributes are non-atomic.

R is not in 2NF, as there are attributes (e.g. CF) that functionally depend on only part (e.g. B) of the full candidate key of the relation (BD).

So, R is currently in 1NF

c) [7 marks] If R is not in 3NF, transform it into a set of 3NF relations.

Answer: Assume FD set is minimal (OK given lecture/tutorial coverage this year)

Pull all non-complete FDs out into own relations, and make sure one relation hold the previously determined key, giving:

R1 (<u>B</u>, <u>D</u>)

R2 (<u>B</u>, C, F)

R3 (<u>D</u>, A, E)

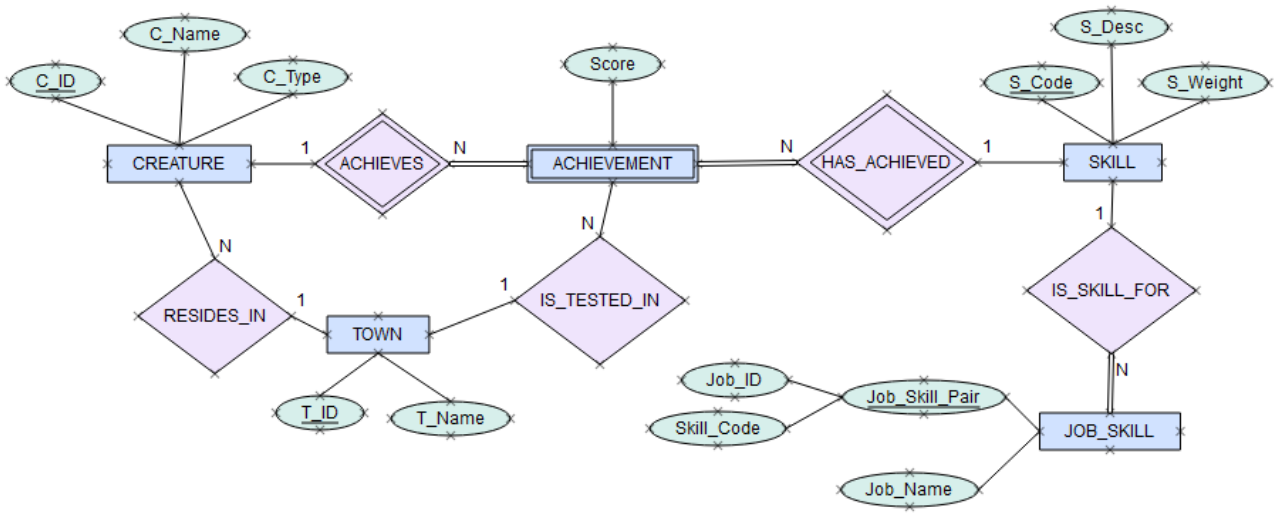
R4 (<u>E</u>, A)

3 [40 marks for the whole question] In the lab test, you worked with a small database storing data for a creature/achievement/skill scenario. The five (slightly modified) tables here have the following structure, with the relational tables and columns named as follows:

```
Creature (C_id, C_name, C_Type, reside_t_id)
                    -- reside_t_id: residence town for that creature
Skill (S_code, S_desc, S_weight)
Achievement (Ach_ID, C_id, S_Code, score, test_t_id)
                    -- test_t_id : testing town for that achievement
Town (T_id, T_name)
JobSkill (Job_ID, Job_Name, S_Code)
```

a) [15 marks] Draw an EER diagram that reflects all tables and relationships from the schema described above.

Preferred Answer:



Could also show Achievement as a strong entity (though issue is either need to create arbitrary Ach_ID or somewhat redundantly add C_ID and S_Code as composite key attributes), or show Achievement as relationship (though then represent town relationships differently for reside town and test town). The above solution deals with these issues mostly cleanly.

b) [4 marks] Write a single SQL statement to find the name of each creature that has had at least one skill achievement that was tested in the town of Rotorua.

Answer:

```
SELECT C_name
FROM Creature C
JOIN Achievement A ON C.C_id = A.C_id
JOIN Town T ON A.test_t_id = T.T_id
WHERE LOWER(T.T_name) = 'rotorua';
```

c) [4 marks] State the result of the following SQL query in English:

```
SELECT DISTINCT T_name
FROM Town T
WHERE T.T_id IN
   (SELECT C.reside_t_id
    FROM Creature C
    JOIN Achievement A ON A.C_id = C.C_id
    WHERE A.test_t_id = C.reside_t_id);
```

Answer: Find each town by name where a creature both resides in that town and has achieved a skill that was tested in the same town.

d) [7 marks] Given an SQL query as follows:

```
SELECT C.C_ID, A.S_Code
FROM Creature C
JOIN Achievement A on C.C_id = A.C_id
WHERE A.Score = 1;
```

Draw relational algebra query trees for two different relational algebra queries corresponding to this SQL query, with one of the two query trees being the canonical tree. Discuss how efficient each query tree would be to execute, in relation to various options for the physical design.

Answer (example):

<image: query tree with Cartesian products to combine tables first, then selects implementing join conditions and score = 1 condition second, and projection of desired attributes third.>. This is a canonical tree, and is less efficient in that executing it combines the full tables.

<image: query tree with select on Achievement for score = 1 first, then join Achievement with Creature on join condition, then project desired attributes last> This is a more efficient tree, as it only joins a subset of the Achievement table to the Creature table.

First solution:

R3 ← Creature X Achievement

R4 ← $\sigma_{\text{Creature.c\_id = Achievement.c\_id and score = 1}}$ (R3)

R5 ← $\pi_{\text{Creature.c\_id, Achievement.s\_code}}$ (R4)

Tree:

$\Pi_{\text{Creature.c\_id, Achievement.s\_code}}$

|

$\Sigma_{\text{Creature.c\_id = Achievement.c\_id and score = 1}}$

|

X

/ \

Creature Achievement

Second solution (many others possible):

$R1 \leftarrow ((\sigma_{score = 1} Achievement) \bowtie_{Creature.c\_id = Achievement.c\_id} Creature)$

$R2 \leftarrow \pi_{Creature.c\_id,\ Achievement.s\_code}\ (R1)$

Tree:

$\pi_{Creature.c\_id,\ Achievement.s\_code}$

|

$\bowtie_{Creature.c\_id = Achievement.c\_id}$

/ \

$\sigma_{score = 1}$ Creature

|

Achievement

The first solution is the canonical tree, as it combines the tables first and then selects and projects later. This first solution will be more expensive than the second solution, as the early Cartesian product in the first solution generates a larger product table than the one generated by combining after filtering out rows and columns.

e) [3 marks] Is the JobSkill table in 3rd Normal Form? Briefly explain why or why not.

Answer: **No, JobSkill is not in 3NF, as contains a partial dependency (Job_Name, which is only dependent on Job_ID, not on the full primary key of both Job_ID and S_code.**

f) [4 marks] Write a single SQL statement to create a new Job table that holds an arbitrary Job_ID (as its key) and the name of the Job. Note that these fields are currently in the JobSkill table.

Answer:

```
CREATE TABLE Job
(
   Job_ID    integer,
   Job_Name varchar(30),
   primary key (Job_ID)
```

);

g) **[3 marks]** IF we populate and use this new Job table (NOTE: you don't have to populate/use it here), what normalization change(s) would we make to our existing tables?

Remove Job_Name field from the JobSkill relation, as that field is now correctly located in the Job relation.

---

4 [18 marks for the whole question]

a) **[4 marks]** Briefly describe why the SQL query you send to the DBMS may not be executed by the DBMS in the same form that you wrote it.

Answer: The query optimization engine develops multiple execution plans based on your original query, and chooses the most efficient plan using a cost-based optimization process. So, the execution plan chosen may not match your original query.

b) **[4 marks]** Why is cost-based query optimization preferable to heuristic query optimization?

Answer: Cost-based optimization is preferable to heuristic query optimization as it uses the experience of information from past queries, the index structure and other factors to customize a specific best solution. Heuristic query optimization only goes through a list of rules to find the best general technique for this data and structure, but does not use actual past data.

c) **[6 marks]** Briefly describe how B+ trees are different than binary trees.

Answer: Binary trees tend to be deep but not wide, leading to longer average access times.

Binary trees can be quite unbalanced, leading to uneven access times.

Binary trees may be difficult to rebalance, and difficult for changes (additions, modifications, deletions) to the tree

Binary trees allow relatively faster search for one item, but are not as easily searched in linear order

B+-trees tend to be shallow and wide with all data at the leaf level, leading to short and consistent access times.

B+-trees remain balanced, leading to even access times

B+ trees can be updated in logarithmic time, making changes easier.

B+ trees have an extra block pointer at the leaf level, leading to easy ordered traversal

d) **[4 marks]** What are the two major types of indexing that are most commonly used for implementing relational DBMS indexes?

Answer: Hashing and B/B+ (multilevel balanced) trees

5 [14 marks for the whole question]

a) [4 marks] What is the purpose of a *transaction* in database systems?

Answer: A transaction is a unit of processing that has all of its component statements executed correctly or not at all. This is essential to maintaining the integrity of the database; e.g. execution of one or more inserts into a child table without a corresponding row also being inserted into a parent table could leave the child rows with meaningless foreign key values.

b) [4 marks] In regard to applying transactions, would most DBMSs be considered *optimistic* (i.e. applies each transaction tentatively, performs an undo if necessary) or *pessimistic* (i.e. holds each transaction tentatively, only applies the transaction if all work is completed and committed)? Briefly justify your answer.

Answer: Optimistic, as correct execution is far more common and likely than execution of a transaction that includes an error.

c) [3 marks] Briefly describe the purpose of a Data Access Object (DAO) when writing software to programmatically access a database system.

A DAO isolates DBMS-specific code into one class, so that code for connections and database queries are not scattered through all of your other classes.

d) [3 marks] Briefly describe how SQL injection can allow malformed input to lead to security issues such as unauthorized access to a database system.

SQL injection sends unusual/malformed query strings from an input interface to be passed to the DBMS engine. Inclusion of special characters such as apostrophes, dashes and other characters significant to SQL potentially change the logical result of the SQL query, including returning rows not intended through the original query.

END OF PAPER