Northeastern University
CS5200 – DBMS
Spring 2025, Derbinsky

Exam 1

Name: Erdun E

Problem	Points
General DBMS Knowledge	8 /10
The Relational Data Model	14 /20
SQL #1	2 /4
SQL #2	34 /36
SQL #3	10 /10
Security	3 /5
Bonus: Password Storage	0 /5
Total	$\boxed{71}$ /85

Instructions

- You will have 60 minutes to complete this exam; do **NOT** begin until instructed to do so.
- You are allowed to use one sheet of 8.5 × 11" paper for reference, as well as the provided SQL reference, but no other resources.
- No electronic devices may be used, including calculators, cell phones, cameras, and computers.
- Please write legibly: what I cannot read, I cannot award credit!

(10 pts.) General DBMS Knowledge

Respond to the questions below.

-2

a) ~~DBMS~~ is the declarative language used to define structure and manipulate data, as well as other objects (e.g., permissions), in a relational database.

SQL – Declarative languages

- Structured Query Language (SQL)
  - Data… definition, manipulation, query

b) For each description below, related to online purchases, enter the single best-matching `ACID` property (you must correctly write the full property name for credit):

Once an order is completed, the customer is able to post a review Durability

A customer's order should never make product inventory negative Consistency

Product inventory does not change if the payment method fails to authorize Atomicity

Until it is purchased, many customers can hold a product in their carts Isolation

(20 pts.) The Relational Data Model

Respond to the questions below.

a) Choose the single item from the right that best matches each item on the left.
Items on the right **may be used more than once** and may refer to Figure 1 below.

- Ordered list of *n* ***attributes*** (columns; degree *n* or *n*-ary)
  Each with a corresponding ***domain*** (set of valid ***atomic*** values)
  - dom(SSN) = "###-##-####"
  - dom(GPA) = [0, 4]

F	Row	A. Attribute
A	Column	B. Domain
B ~~X~~	Permissible atomic value(s) for a column	C. Foreign Key
E	Table	D. NULL
E ~~X~~	Table state: set/bag of ...	E. Relation
D	Entity Integrity: *invalid* value of primary key	F. Tuple
D	Unknown/not available	
C ~~X~~	Mechanism of referential integrity	
B	Explicit constraint: value must be within [0, 4]	
E	Figure 1: Foo	
A	Figure 1: c	
C	Figure 1: arrow	

The Correct answer is B that Domain, because the domain defines the set of permissible atomic values for a column. And it restricts what kind of data can be stored in a column

The Correct answer is E that Relation, because a relation represents a table in a relational database. The state of a relation is the set or bag of tuples in that table.

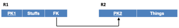
A database consists of…
i. a set of ***relations*** (tables)
ii. a set of ***integrity constraints***

The Correct answer is C that Relation, because it ensures that the values in a column in one table match values in the primary key of another table. This prevents invalid references between tables.

Referential Integrity

All tuples in relation R1 must reference an existing tuple in relation R2 (R1 *may* be the same as R2)

A **foreign key** (FK) in R1 references R2 iff…
- The attribute(s) in FK have the same domain(s) as the primary key attribute(s) PK of R2
- A value of FK in a tuple t1 either is NULL or occurs as a value of PK for some tuple t2 (t1 *refers to* t2)



-3

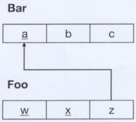


Figure 1: Relational Schema

b) Indicate the validity of each of the following statements by writing the complete word `true` or `false`.

False ~~True~~ A table's schema dictates how rows are ordered.

True In a real database table without any keys, two rows can have the same values for all columns.

False Because of the many features, a Relational Database Management System should *always* be used to manage an application's user data.

-1

{z},{x,y}

c) List all potential primary keys for the current state of `Baz`: {z}. Missing!
(express each response as a set {}).

A relation may have multiple keys (each is a **candidate key**). Relations commonly have a **primary key** (underlined, PK; typically small number of attributes, used to *identify* tuples), and may also have some number of additional **unique key(s)**.

Tuples: Theory vs. Implementation

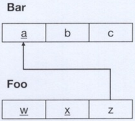
- Relation state is formally defined as a *set* of tuples, implying…
  - No inherent order
  - No duplicates
- In real database systems, the rows on disk will have an ordering, but the relation definition sets no preference as to this ordering
  - We will discuss later in physical design how to establish an ordering to improve query efficiency

Baz

w	x	y	z
1	a	α	vi
2	b	α	v
3	c	α	iv
4	a	β	iii
5	b	β	ii
NULL	c	β	i

(4 pts.) SQL #1

Consider the following relational schema reproduced from Figure 1.



Furthermore, assume...

- Bar has **10 rows**
- Foo has **50 rows** and NULL is not a permissible value of z

Characterize the result of the query...

```
SELECT * FROM Foo f INNER JOIN Bar b ON f.z=b.a
```

by indicating the number of...

[INNER] JOIN $A \bowtie B$	Row must exist in both tables

6 columns

~~60~~ rows -1

50 Rows

Because From Foo and Foo has 50 rows

Inner join will join 0 extra rows because no exist row in both table

and a brief description why...

6 columns = Foo has 3 cols, Bar has 3 cols

60 rows = Foo has 50 rows, ~~Bar has 10 rows~~

? -1

(36 pts.) SQL #2

Consider the following database consisting of the **Users** *and* **AppRatings** *tables.*

Users

id	name	age
1	Alice	30
2	Bob	22
3	Cathy	50
4	Dylan	18

AppRatings

user	app	rating
1	Wordle	2
1	Spelling Bee	5
1	Connections	5
2	Wordle	3
2	Spelling Bee	5
3	Wordle	4

a) Find 4 errors in the DDL code to build the above database.

- Circle each error and label it with a number (1–4)
- In the corresponding line below, describe the problem

```
CREATE TABLE Users (
    id INT,
    name VARCHAR(10)     1
);

CREATE TABLE AppRatings (
    user INT PRIMARY KEY,
    app VARCHAR(20),       2
    rating INT PRIMARY KEY,    3
    FOREIGN KEY (id) REFERENCES Users (user)    4
);
```

Yes, id is the primary key in table Users but there didn't add constranits for it as a primary key

1. didn't define the constraint to the age +2
it's should be like age Int

2. app is primary key, but there don't add constriants for it
one issue +2

3. rating is not primary key. should delete it.

4. Wrong input. FOREIGN KEY (user) REFERENCES Users (id)
+2

b) Draw the exact result produced from the following query:

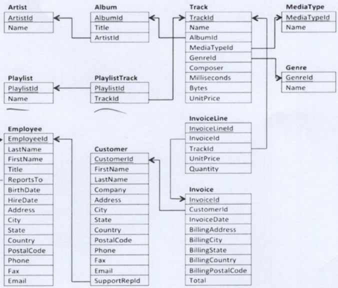
```
SELECT
   ar.app AS a,
   AVG(ar.rating) AS b,
   MAX(u.age) AS c,
   COUNT(*) AS d
FROM
   Users u INNER JOIN AppRatings ar ON u.id=ar.user
GROUP BY
   ar.app
ORDER BY
   b DESC,
   d DESC,
   a ASC
```

a	b	c	d
Spelling Bee	5	30	2
Connections	5	30	1
Wordle	3	50	3

−0

(10 pts.) SQL #3

Write an SQL query (valid in SQLite) against the Chinook database according to the prompt.



Multiple playlists contain the word "Classical" – determine the total number of tracks across all of them. (Note: a track can occur on multiple playlists and, if so, should be tallied multiple times.)

numClassical
150

```
Select
    Count(Track.TrackID) AS numClassical
From
    Track  ← not needed, but fine
    Inner Join PlaylistTrack on Track.TrackID = PlaylistTrack.TrackID
    Inner Join Playlist on PlaylistTrack.playlistID = Playlist.PlaylistID
Where
    Playlist.Name LIKE '% Classical %'
```

−0

- Key idea: store multiple salted/hashed passwords for each user
  - As usual, users create a single password and use it to login
  - User is unaware that additional honeywords are stored with their account
- Implement a honeyserver that stores the index of the correct password for each user
  - Honeyserver is logically and physically separate from the password database
  - Silently checks that users are logging in with true passwords, not honeywords
- What happens after a data breach?
  - Attacker dumps the user/password database…
  - But the attacker doesn't know which passwords are honeywords
  - Attacker cracks all passwords and uses them to login to accounts
  - If the attacker logs-in with a honeyword, the honeyserver raises an alert!

(5 pts.) Security

Respond to the questions below.

Honeywords are a system-generated technique to confuse attackers by storing fake passwords alongside real ones. Users are not required to select "clever" passwords for this system to work.

Indicate the validity of each of the following statements by writing the complete word `true` or `false`.

False ~~True~~ — Users are encouraged to select clever honeywords to prevent data breaches.

False ~~True~~ — The `md5` hash function is considered effective for protecting sensitive information.

False — It is safe to store passwords in plain text if a user-selected number ("salt") is also stored in plain text.

True — Data from past data breaches indicates that user-selected passwords are not random, which commonly makes dictionary attacks effective.

False — Manually quoting user-input values is the safest protection against SQL injection attacks.

MD5 is no longer considered effective for protecting sensitive information due to its vulnerability to collision attacks. Recommend using bcrypt, PBKDF2, or scrypt instead for secure password hashing.

Hashed Passwords

- Key idea: store encrypted versions of passwords
  - Use one-way cryptographic hash functions
  - Examples: MD5, SHA1, SHA256, SHA512, bcrypt, PBKDF2, scrypt
- Cryptographic hash function transform input data into scrambled output data
  - Deterministic: hash(A) = hash(A)
  - High entropy:
    - MD5('security') = e91e6348157868de9dd8b25c81aebfb9
    - MD5('security1') = 8632c375e9eba096df51844a5a43ae93
    - MD5('Security') = 2fae32629d4ef4fc6341f1751b405e45
  - Collision resistant
    - Locating A' such that hash(A) = hash(A') takes a long time (hopefully)
    - Example: 221 tries for md5

(5 pts.) Bonus: Password Storage

Respond to the questions below.

Consider a user-login table that contains a password field. Now assume a well-intentioned database developer has just learned about secure password storage and so decides to append to each password, prior to hashing, a single randomly generated salt value (that is, one salt for the entire table, such as 42). Finally, assume an attacker gains access to this table of hashes. Answer the following questions related to password cracking in this scenario.

a) Explain the effect of this type of approach on preventing attack.

If use one salt for the entire table, it's better than having no salt at all because it stops precomputed attacks like rainbow tables. But the issue is if the attacker gets access to both the table and the salt, they can still compute the hashes for all the passwords at once. So, it's not as strong as giving each password its own unique salt.

Hardening Password Hashes

- Key problem: cryptographic hashes are deterministic
  - hash('p4ssw0rd') = hash('p4ssw0rd')
  - This enables attackers to build lists of hashes
- Solution: make each password hash unique
  - Add a salt to each password before hashing
  - hash(salt + password) = password hash
  - Each user has a unique, random salt
  - Salts can be stores in plain text

b) Now assume the attacker has successfully cracked a few passwords and inspects the results. How may the attacker more efficiently crack the remaining hashes.

Once the attacker cracks a few passwords, they'll look for patterns, maybe a lot of users have weak or common passwords. Since the whole table uses the same salt, they can just reuse that salt and quickly try dictionary-based attacks for the remaining passwords.

Attacking Password Hashes

- Recall: cryptographic hashes are collision resistant
  - Locating A' such that hash(A) = hash(A') takes a long time (hopefully)
- Are hashed password secure from cracking?
  - No!
- Problem: users choose poor passwords
  - Most common passwords: 123456, password
  - Username: cbw, Password: cbw
- Weak passwords enable dictionary attacks

Dictionary Attacks

- Common for 60-70% of hashed passwords to be cracked in <24 hours

c) Imagine you are newly hired as a database developer, and come across this salting technique. How do you improve this hashing policy? How will this impact current users of the system?

1. Use a unique salt for every password. That way, even if two users have the same password, their hashes will still look different.
2. Switch to a modern hashing algorithm, like bcrypt or PBKDF2, which are much harder to brute force.
3. Regularly update your hashing methods to stay ahead of attackers.

Northeastern University CS5200 – DBMS • Spring 2025 • Derbinsky

Hardening Password Hashes

- Key problem: cryptographic hashes are deterministic
  - hash('p4ssw0rd') = hash('p4ssw0rd')
  - This enables attackers to build lists of hashes
- Solution: make each password hash unique
  - Add a salt to each password before hashing
  - hash(salt + password) = password hash
  - Each user has a unique, random salt
  - Salts can be stores in plain text

Security

Hardening Salted Passwords

- Problem: typical hashing algorithms are too fast
  - Enables GPUs to brute-force passwords
- Old solution: hash the password multiple times
  - Known as **key stretching**
  - Example: crypt used 25 rounds of DES
- New solution: use hash functions that are designed to be slow
  - Examples: bcrypt, PBKDF2, scrypt
  - These algorithms include a work factor that increases the time complexity of the calculation
  - scrypt also requires a large amount of memory to compute, further complicating brute-force attacks