Midterm Review

Matt Warner

Chapter 1

Overview

1 Vocabulary

- You're responsible for knowing the terms used in the lectures.
- There will be some questions specifically on vocabulary.
- Other questions will use the terms as part of a larger question - they will not be redefined during the test.

2 ER Diagrams

- Be able to read and interpret them.
- Provide cardinalities/connectivities.
- Degree of a relationship.
- Recursive relationships.
- Inheritance - specialization/generalization, overlapping/disjoing subtypes.

3 Logical Data Model - Relational Model

- What makes up the relational model?
- Relations, tuples, attributes, etc.
- Keys - primary key, candidate key, superkey.
- Know the requirements for each type of key.
- How do the above keys work with respect to functional dependencies?
- Foreign Keys - what are they and how do they work?
- Entity Integrity Constraint.
- Referential Integrity Constraint.
- Conversion of ER diagrams to ${}_3NF$ relations.

4 Normalization

- Be able to identify and fix violations of all 3
  - 1_NF - no repeating groups, all values must be atomic.
  - 2_NF - No partial key dependencies.
  - $_3NF$ - No transitive dependencies.

5 SQL

- Know the commands and be able to use them to make queries.
  - DDL - `CREATE TABLE, ALTER TABLE, DROP TABLE, (DESCRIBE)`
  - DML - `INSERT, UPDATE, DELETE, SELECT`
- Be sure to know which commands are DDL vs . DML
- `LIKE operator`
- `ORDER BY`
- `DISTINCT`
- `GROUP BY`
- Group functions
- Subqueries
  - How they get evaluated.
  - Multi-rpw
  - Single-value

Chapter 2

Vocabulary

1 Databases

Generic Database terms:

- Enterprise - A generic term for any reasonably large-scale commerical, scientific, technical, or other application.
- Operational data - Data maintained about the operation of an enterprise. Stuff like products, accounts, patients, students, plans. DOES NOT INCLUDE input/output data.
- DBMS - A collection of programs that enables users to create and maintain a database, this collection of programs forms a general-purpose software system that facilitates
  - Definition of databases.
  - Construction of databases.
  - manipulation of data within a database.
  - Sharing of data between users/applications.

Capabilities of a DBMS (Terms)

Transaction management - A feature that provides correct, concurrent access to the database, possibly by many users at the same time.

Access Control - The ability to limit access to data by unauthorized users along with the capability to check the validity of the data.

Resiliency - The ability to recover from system failures without losing data.

Leveled Architecture of a DBMS(Terms)

External Level - A view or sub-schema. This is a portion of the logical database.

Logical Level - Abstraction of the real world as it pertains to the users of the database.

Physical Level - The collection of files and indicies stored on secondary storage device (HDD, SSD, etc.). This is the actual data.

Basic Database Terminology

Instance - An instance of the database is the actual contents of the data. An instance of the database could be an extension of the database, the current state of the database, or even a snapshot of the data at a given point in time.

Schema - The schema of a database a description of what data can be stored. This can be though of as the data members of a class. We have the names, which tell us what what the values are going to be but we dont have the values.

Data Independence - A property of an appropriately designed database system. This has to do with the Leveled Architecture, and the mapping of logical level to physical level, and logical to external.

- **Physical data independence** - Physical schema can be changed without modifying logical schema.
- **Logical data independence** - Logical schema can be changed without having to modify any of the external views.

DCL - The Data control language.

DDL - Data definition language.

DMl - Data manipulation language.

2 ER Diagrams

Data Models

Data Models - A means of describing the structure of data. An ER Diagram is a conceptual data model, but we can also have logical data models (relational, network, hierarchical, inverted list, or object-oriented). The logical data model used in this course is the relational model.

- **Conceptual Data Model** - Shows the structure of the data including how things are related. This is a communication tool and is independent of comericial DBMSes. They are relatively easy to learn and use and help show the semantics or meaning of the data.
- **Logical Data Models** - Data is stored in relations (tables). These tables have one value per cell. Commercial relational data models include DB_2, Oracle, Ingress, MySQL, Microsoft Access.

ERD components

Entities - Principle objects about which information is kept - These are "things" we store data about. If you were to look at the ER Diagram like a spoken language, the entities are nouns - Person, Place, thing, event.

Relationships - Relationships connect on ore more entities together to show an association. A relationship cannot exist without at least one associated entity. Represented as diamonds.

Attributes - Characterists of entities or of relationships. These represent some small piece of associated data. Represented by either a rounded rectangle or an oval.

- Attributes on Entities -

Chapter 3

Relational Model

1 Keys

Superkey

A super key is an attribute or set of attributes whose values can uniquely identify any tuple within that relation. Every relation has at least one - the set of all attributes in the relation (since duplicate tuples are considered to be the same tuple)

Candidate key

A minimal super key - the minimum set of attributes necessary to uniquely identify a tuple within the relation.

Primary key

The primary key for a relation is chosen from among the relations candidate keys. It becomes the official key that is used to reference tuples within the relation. There can only be one.

Once a primary key is chosen, each of the attributes in the relation will be either prime or non-prime.

2 Requirements for keys

3 How do the above keys work with respect to function dependencies?

4 Foreign keys - what are they and how do they work?

5 Referential Integrity Constraint

This applies to all foreign keys.

It constrains the values of foreign keys in relations to values that acutally exist as primary keys for tuples within the home relation.

If the foreign key is otherwise allowed to be NULL, then that is also an acceptable value.

Chapter 4

SQL

1 DDL

1.1 creating a table

This is the basic format of a `CREATE TABLE` statement.

```
CREATE TABLE <table_name> (
    <attribute>  <type> [NOT NULL] [UNIQUE] [PRIMARY KEY], [...]
    [PRIMARY KEY(<pkattrs>),]
    [FOREIGN KEY(<attr_here>) REFERENCES <home_table>(<attr_home>)]
)
```

Placeholder	Purpose
<table_name>	name of the table
<attribute>	name of the attribute
<type>	data type of the current attribute
<pkattrs>	comma seperated list of the attributes makeing up the table's primary key.
<attr_here>	comma-seperated list of attributes in the current table forming a foreign key.
<home_table>	name of the home table
<attr_home >	comma-seperated list of attributes in the home table, matching the attributesin <attr_here>

Heres an example:

```
CREATE TABLE Dog (
    dog_id UNSIGNED SMALLINT [AUTO_INCREMENT],
    name VARCHAR(20),
    weight FLOAT,
    -- Primary key and foreign keys placed here (makes things more clear).
    PRIMARY KEY(dog_id)
);
```

Setting the primary key

There are two ways to set the primary key:

- For single attribute primary keys, you can use the PRIMARY KEY column option. THe option may only be used once, and proclaims that the single attribute is the **entirety** of the primary key.
- If you have multiple attributes in the primary key, the only way is to add the seperate constraint:

  ```
  -- set up primary key with more than one attribute (also works with single attr).
  PRIMARY KEY(<x>, <y>, <z>, <etc>)
  ```

Setting the foreign key

A foreign key links the current table to another table, which we call the home relation.

- The foreign key must contain all of the attributes of the primary key of the home relation.
- They may have different names in each of the tables, but there needs to be a match for each.
- Each of these attributes must have the exact same data type as its counterpart in the home table.

If a table is to contain a foreign key, we include a constraint in our CREATE TABLE statement like the following

```
FOREIGN KEY (<localnames>) REFERENCES <home_table>(<homenames>)
```

This can be done for multiple foreign keys, filling in the placeholders <localnames>, <home_table>, and <homenames>appropriately for each.

Say we had: ***Student***(SSN†, CLSYEAR, GPA, TOTALHRS)

```
CREATE TABLE Student (
SSN CHAR(9), NOT NULL,
CLSYEAR CHAR(9),
GPA DECIMAL(4.3),
TOTALHRS INT,

PRIMARY KEY (SSN), -- Set up the primary key seperately
FOREIGN KEY (SSN) REFERENCES Person(SSN) -- a Student is a Person
```

1.2 Altering a table

An ALTER TABLE statement will allow you to have the DBMS make changes to the schema of a table that has already been created. It works with various subcommands.

- ALTER TABLE ADD
- ALTER TABLE MODIFY
- ALTER TABLE DROP

ALTER TABLE ADD

The ALTER TABLE ADD command can be used to add a new column or new columns to the schema of an existing table.

Adding a single column/attribute:

```
ALTER TABLE <table_name> ADD <attribute> <type>;
```

Adding multiple columns/attributes:

```
ALTER TABLE <table_name> ADD (<attribute> <type>, ...);
```

Where <attribute>and <type>have the same meaning as they did with CREATE TABLE. For example, using the Person table that we created eariler, let's say we'd like to start storing people's birthdays. We can use the command

```
ALTER TABLE ADD Birthday DATE;
```

ALTER TABLE MODIFY

The ALTER TABLE MODIFY command can be used to change properties of a column/attribute (including type, length, and other column options) in a table that already exists.

```
ALTER TABLE MODIFY <col_name> <new_options>;
```

Lets say we run into some people that have last names longer than 20 characters. In order to be able to store their names without truncation, we need to change the data type for the LNAME column. We can do that with the command:

```
ALTER TABLE Person MODIFY LNAME CHAR(32);
```

ALTER TABLE DROP

The ALTER TABLE DROP command can be used to remove a column/attribute from the *schema* of a table.

```
ALTER TABLE DROP <col_name>;
```

Let's say we dont want to store information on birthdays anymore. We can use the command:

```
ALTER TABLE Person DROP Birthday;
```

to get rid of the Birthday column/attribute we added to the table eariler.

1.3 DROP TABLE

To remove a table from the database, we can use the DROP TABLE command.

```
DROP TABLE <table>;
Note that you cant drop a table if its primary key is used as a foreign key in another
↪  table.
```

2 DML

The Data Manipulation Language (DML) is the language used to work with the instance data. In SQL, this means doing things with the rows contained by tables, rather than to the tables themselves.

Here is the table showing the statements that are part of the DDL in SQL, along with a short description of their function.

Statement	Function
INSERT	add a new row to a table
UPDATE	change values in an existing row
DELETE	remove rows from the table
SELECT	display the data stored in rows (details covered later)

2.1 INSERT

```
INSERT INTO <table_name>
    VALUES (<value_list>);

INSERT INTO <table_name>
    (<attr_list>)
    VALUES (<value_list>)

INSERT INTO <table_name>
    <another_query>;
```

Let's add a row into the Person table. We'll use the initial version with the four columns we created it with.

```
INSERT INTO Person
    VALUES('123456789',
           'Inigo',
           'Montoya',
           '55555555');

-- Method 2
INSERT INTO Person
    (LNAME, FNAME, SSN)
```

```
        VALUES('Buttercup',
               'Princess',
               '987654321');
        )
```

columns not in the attribute list are set to their default value if possible. This is why PHONE is NULL. This version of the INSERT statement is better if you're making SQL that needs to be in a script that is to be run later, as it tolerates more changes to the table schema than the other version.

The WHERE clause

When working with DML statements, it will be desirable to be able to work only with specific rows. This can be accomplished using a WHERE clause.

The where clause is the keyword WHERE followed by an expression that evaluates to either true or false. It is included in an SQL query to control which rows are affected by the query.

```
WHERE <expression>
```

The expression after WHERE is evaluated one time per row.

Rows where the expression evaluates as true are included in the operation.

Rows where the expression evaluates to false are excluded from the operation.

WHERE clauses are generally used in UPDATE, DELETE, and SELECT statements.

2.2 UPDATE

```
UPDATE <table_name>
SET <attr> = <value> [, <attr> = <value> ...]
[WHERE <expression>]

-- Example
UPDATE Student
Set CLSYEAR = 'Senior'
WHERE TOTALHRS > 90;
```

This has the potential to change many rows, but only the rows with a value for TOTALHRS that is greater than 90.

Lets say that you've somehow gained access to the dattabase that stores grades for the university. Coincidentally, it looks like the table you created in a 466 example! You partied way too hard last semester, so you want to change your GPA (and only your GPA) to the best one possible. If your SSN where 9999999, then the SQL statement.

```
UPDATE STUDENT
SET GPA = 4.000
WHERE SSN = '9999999';
```

Would change your GPA to 4.000 and leave everyone else's alone.

If you want to guarantee that you'll affect only a single row, you'll need to use an attribute that you know is unique. In general, the primary key is the best tool for this, which is why we used the SSN column in the example.

Heres an example of using UPDATE on multiple attributes.

```
-- No WHERE clauses, affects the whole table.
UPDATE Person
    SET FNAME = 'Spartacus',
    LNAME = ' ',
    PHONE = ' ',
```

2.3 DELETE

To delete the rows without getting rid of the table, use a DELETE statement.

```
DELETE FROM <table_name>
[ WHERE <expression> ]
```

Its important to realize that all rows are affected by default, so if a WHERE clause is not supplied, all of the rows will be deleted.

```
DELETE FROM Person
WHERE SSN='123456789';
```

If we want to remove a specific tuple, we need to use a unique column/columns in our WHERE clause. The primary key will always work.

Since all rows are affected by default, failure to specify a WHERE clause will cause all of the rows of the table to be deleted

```
-- Deletes every row in the Student table.
DELETE FROM Student;
```

3 The SELECT Statement

3.0.1 SELECT STatement Format

Two version of the basic format of a SELECT statement follow.

```
SELECT [DISTINCT|ALL] <column_list>
FROM <table_list>
[WHERE <where_expr>]
[GROUP BY <group_key>]
[HAVING <having_expr]
[ORDER BY <sortcols>] ;

SELECT <anyexpression>
```

Get supplier numbers and status for suppliers in Paris.

```
SELECT S, STATUS
FROM S
WHERE CITY = 'Paris';
```

Get part numbers for all parts suplied SELECT P FROM SP;

3.1 ORDER BY

We can use ORDER BY clause to enforce a sorting order on the results of our select statement.

```
ORDER BY <attrs>
```

Example:

```
SELECT S, STATUS
FROM S
WHERE CITY = 'Paris'
ORDER BY STATUS DESC -- sort based on STATUS in descending order
```