MIS2502:
Review for Exam 1

Jing Gong

gong@temple.edu

http://community.mis.temple.edu/gong
Overview

• **Date/Time:** Tuesday, Feb. 16, in class (1 hour 20 minutes)
• **Place:** Regular classroom

Please arrive 5 minutes early!

• Multiple-choice and short-answer questions
• Closed-book, closed-note
• No computer
Coverage

Check the Exam 1 Study Guide

Not every item on this list may be on the exam, and there may be items on the exam not on this list.
The Things You Can Do With Data/ The Information Architecture of an Organization

• Data vs. information

• Transaction
  – Business transaction
  – Database transaction

• Transactional database vs. Analytical data store
  – Characteristics, goals, relationship
The Information Architecture of an Organization

- **Transactional Database**: Stores real-time transactional data.
  - Called OLTP: Online transaction processing.

- **Analytical Data Store**: Stores historical transactional and summary data.
  - Called OLAP: Online analytical processing.

Data entry → Transactional Database → Data extraction → Analytical Data Store → Data analysis
Comparing Transactional and Analytical Data Stores

<table>
<thead>
<tr>
<th>Transactional Database</th>
<th>Analytical Data Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Based on Relational paradigm</td>
<td>Based on Dimensional paradigm</td>
</tr>
<tr>
<td>Storage of real-time transactional data</td>
<td>Storage of historical transactional data</td>
</tr>
<tr>
<td>Optimized for storage efficiency and data integrity</td>
<td>Optimized for data retrieval and summarization</td>
</tr>
<tr>
<td>Supports day-to-day operations</td>
<td>Supports periodic and on-demand analysis</td>
</tr>
</tbody>
</table>
Relational Data Modeling

• Be able to interpret simple ERDs
• Draw an ERD based on a scenario description
  – Entities
  – Relationships & Cardinality
    • Maximum cardinality: One-to-one, one-to-many, many-to-many
    • Minimum cardinality: optional or mandatory (i.e., 0 or 1)
  – Attributes
    • Entity attributes vs. relationship attributes
    • Primary key and non-key attributes
• Best practices for normalization
Cardinality: Example 1

• Each department has many faculty members

• But each faculty is affiliated with one and only one department

This example is part of the “Additional ERD and Schema Exercise”
Cardinality: Example 2

Each student needs to declare **at least one** and **at most two** majors (that is, first major and second major) at the beginning of the first year.
Relational Data Modeling

• Draw the corresponding schema of an ERD

1. Create a table for every entity
2. Create table fields for every entity’s attributes
3. Implement relationships between the tables

1:many relationships
- Primary key field of “1” table put into “many” table as foreign key field

many:many relationships
- Create new table
  - 1:many relationships with original tables

1:1 relationships
- Primary key field of one table put into other table as foreign key field

• Given data from two tables, be able to draw the results of a join of those tables with the data correctly “matched up”
Make the Schema: Example 1

- **Department**
  - DepartmentID
  - DepartmentName

- **Faculty**
  - FacultyID
  - LastName
  - FirstName
  - HighestDegree
  - Title
  - Email

- **Has** relationship
  - DepartmentID
  - FacultyID
Make the Schema: Example 2

Major
- MajorID
- Name
- Description

MajorID

Declares
- First/Second

Student-Major
- StudentMajorID
- StudentID
- MajorID
- First/Second

Student
- StudentID
- LastName
- FirstName
- Email
- ClassYear
Basic SQL

• SELECTing from a single table will be on this exam

• JOINS, subselects, etc. will be on the second exam

• Creating, updating, and deleting tables and rows will also be on the second exam
Basic SQL

• Given a schema of a database, be able to create a SQL statement to answer a question

• Understand how to use
  – SELECT
  – FROM
  – DISTINCT
  – WHERE (and how to specify conditions)
  – AND/OR
  – COUNT, AVG, MIN, MAX, SUM
  – GROUP BY
  – ORDER BY (ASC/DESC)
SELECT from 1 Table

Recall the Customer table...

<table>
<thead>
<tr>
<th>CustomerID</th>
<th>FirstName</th>
<th>LastName</th>
<th>City</th>
<th>State</th>
<th>Zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Greg</td>
<td>House</td>
<td>Princeton</td>
<td>NJ</td>
<td>09120</td>
</tr>
<tr>
<td>1002</td>
<td>Lisa</td>
<td>Cuddy</td>
<td>Plainsboro</td>
<td>NJ</td>
<td>09123</td>
</tr>
<tr>
<td>1003</td>
<td>James</td>
<td>Wilson</td>
<td>Pittsgrove</td>
<td>NJ</td>
<td>09121</td>
</tr>
<tr>
<td>1004</td>
<td>Eric</td>
<td>Foreman</td>
<td>Warminster</td>
<td>PA</td>
<td>19111</td>
</tr>
</tbody>
</table>

SELECT *  
FROM orderdb.Customer;

SELECT FirstName, LastName 
FROM orderdb.Customer 
WHERE State='NJ';
Order By, Group By

• GROUP BY:
  – Return the results for each group in the specified field
  – Often used together with the functions
    – COUNT, MIN, MAX, AVG, SUM
  – One result is returned for each group

• ORDER BY:
  – Sort records based on the specified field or function
  – Default: ASC
  – Specify DESC to make it descending.
Ask: What is the total quantity sold per order (by order number)?

SELECT OrderNumber, SUM(Quantity) FROM orderdb.`Order-Product` GROUP BY OrderNumber;
Group By and Order By (Example)

Ask: What is the total quantity sold per order (by order number)?

Sort the results descendingly based on total quantity sold

```sql
SELECT OrderNumber, SUM(Quantity) 
FROM  orderdb.`Order-Product`
GROUP BY OrderNumber
ORDER BY SUM(Quantity) DESC;
```
DISTINCT and GROUP BY

SELECT DISTINCT State
FROM orderdb.Customer;

SELECT State
FROM orderdb.Customer
GROUP BY State;

TIP: GROUP BY can be used without aggregation functions.
In this case, the result returned will be similar to DISTINCT
Combining WHERE, GROUP BY, and ORDER BY

Recall the Customer table:

<table>
<thead>
<tr>
<th>CustomerID</th>
<th>FirstName</th>
<th>LastName</th>
<th>City</th>
<th>State</th>
<th>Zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Greg</td>
<td>House</td>
<td>Princeton</td>
<td>NJ</td>
<td>09120</td>
</tr>
<tr>
<td>1002</td>
<td>Lisa</td>
<td>Cuddy</td>
<td>Plainsboro</td>
<td>NJ</td>
<td>09123</td>
</tr>
<tr>
<td>1003</td>
<td>James</td>
<td>Wilson</td>
<td>Pittsgrove</td>
<td>NJ</td>
<td>09121</td>
</tr>
<tr>
<td>1004</td>
<td>Eric</td>
<td>Foreman</td>
<td>Warminster</td>
<td>PA</td>
<td>19111</td>
</tr>
</tbody>
</table>

Ask:
How many customers are there in each city in New Jersey? Order the results alphabetically in descending order by city.
One more note: Combining WHERE, GROUP BY, and ORDER BY

```
SELECT City, COUNT(*)
FROM orderdb.Customer
WHERE State='NJ'
GROUP BY City
ORDER BY City DESC;
```

This is the correct SQL statement

```
SELECT City, COUNT(*)
FROM orderdb.Customer
GROUP BY City
ORDER BY City DESC
WHERE State='NJ';
```

This won’t work

When combining WHERE, GROUP BY, and ORDER BY, write the WHERE condition first, then GROUP BY, then ORDER BY.
Good Luck!