CMSC424: Database Design

Instructor: Amol Deshpande
amol@cs.umd.edu
Today’s Plan

- SQL: Integrity Constraints, Triggers, Transactions, Roles
- Discuss reading assignment topics
  - E/R Models
- New topics to discuss
  - More constructs in E/R modeling
  - Converting from E/R to relational schema

- Other things
  - Small change in the late days policy (max 4 days for any assignment instead of 5, so I can discuss in class on Wed)
  - DSS: Let me know soon
Outline

- More SQL
  - Views
  - Integrity Constraints
  - Triggers
  - Transactions
  - Authorization
Transactions

- A transaction is a sequence of queries and update statements executed as a single unit
  - Transactions are started implicitly and terminated by one of
    - **commit work**: makes all updates of the transaction permanent in the database
    - **rollback work**: undoes all updates performed by the transaction.

- Motivating example
  - Transfer of money from one account to another involves two steps:
    - deduct from one account and credit to another
  - If one steps succeeds and the other fails, database is in an inconsistent state
  - Therefore, either both steps should succeed or neither should

- If any step of a transaction fails, all work done by the transaction can be undone by rollback work.
- Rollback of incomplete transactions is done automatically, in case of system failures
Transactions (Cont.)

- In most database systems, each SQL statement that executes successfully is automatically committed.
  - Each transaction would then consist of only a single statement.
  - Automatic commit can usually be turned off, allowing multi-statement transactions, but how to do so depends on the database system.
  - Another option in SQL:1999: enclose statements within `begin atomic` ...
    ```sql
    end
    ```
Outline

- More SQL
  - Views
  - Integrity Constraints
  - Triggers
  - Transactions
  - Authorization
A **trigger** is a statement that is executed automatically by the system as a side effect of a modification to the database.

Suppose that instead of allowing negative account balances, the bank deals with overdrafts by

1. setting the account balance to zero
2. creating a loan in the amount of the overdraft
3. giving this loan a loan number identical to the account number of the overdrawn account
create trigger overdraft-trigger after update on account
  referencing new row as nrow
  for each row
  when nrow.balance < 0
  begin atomic
      actions to be taken
  end
create trigger overdraft-trigger after update on account
referencing new row as nrow
for each row
when nrow.balance < 0
begin atomic
  insert into borrower
    (select customer-name, account-number
     from depositor
     where nrow.account-number = depositor.account-number);
  insert into loan values
    (nrow.account-number, nrow.branch-name, nrow.balance);
  update account set balance = 0
  where account.account-number = nrow.account-number
end
Triggers...

-External World Actions
  - How does the DB *order* something if the inventory is low?

-Syntax
  - Every system has its own syntax

-Careful with triggers
  - Cascading triggers, Infinite Sequences...

-More Info/Examples:
  - Google: “create trigger” oracle download-uk
Outline

- More SQL
  - Views
  - Integrity Constraints
  - Triggers
  - Transactions
  - Authorization
IC’s

- Predicates on the database
- Must always be true (checked whenever db gets updated)

There are the following 4 types of IC’s:

- **Key constraints** (1 table)
  - e.g., 2 accts can’t share the same acct_no
- **Attribute constraints** (1 table)
  - e.g., accts must have nonnegative balance
- **Referential Integrity constraints** (2 tables)
  - E.g. bnames associated w/ loans must be names of real branches
- **Global Constraints** (n tables)
  - E.g., all loans must be carried by at least 1 customer with a savings acct
Key Constraints

Idea: specifies that a relation is a set, not a bag

SQL examples:

1. **Primary Key:**
   
   CREATE TABLE branch(
   bname CHAR(15) PRIMARY KEY,
   bcity CHAR(20),
   assets INT);
   
   or
   
   CREATE TABLE depositor(
   cname CHAR(15),
   acct_no CHAR(5),
   PRIMARY KEY(cname, acct_no));

2. **Candidate Keys:**
   
   CREATE TABLE customer ( 
   ssn CHAR(9) PRIMARY KEY, 
   cname CHAR(15), 
   address CHAR(30), 
   city CHAR(10), 
   UNIQUE (cname, address, city));
Key Constraints

Effect of SQL Key declarations
  PRIMARY (A1, A2, .., An) or
  UNIQUE (A1, A2, ..., An)

Insertions: check if any tuple has same values for A1, A2, .., An as any inserted tuple. If found, reject insertion

Updates to any of A1, A2, ..., An: treat as insertion of entire tuple

Primary vs Unique (candidate)
  1. 1 primary key per table, several unique keys allowed.
  2. Only primary key can be referenced by “foreign key” (ref integrity)
  3. DBMS may treat primary key differently
     (e.g.: create an index on PK)

How would you implement something like this?
Attribute Constraints

Idea:
- Attach constraints to values of attributes
- Enhances types system (e.g.: \( \geq 0 \) rather than integer)

In SQL:

1. **NOT NULL**
   - e.g.: `CREATE TABLE branch(
          bname CHAR(15) NOT NULL,
          ....
       )`
   - Note: declaring bname as primary key also prevents null values

2. **CHECK**
   - e.g.: `CREATE TABLE depositor(
          ....
          balance int NOT NULL,
          CHECK( balance \( \geq 0 \)),
          ....
       )`
   - affect insertions, update in affected columns
Attributes Constraints

Domains: can associate constraints with DOMAINS rather than attributes

E.g.: instead of:

```sql
CREATE TABLE depositor(
    ....
    balance INT NOT NULL,
    CHECK (balance >= 0)
)
```

One can write:

```sql
CREATE DOMAIN bank-balance INT (
    CONSTRAINT not-overdrawn CHECK (value >= 0),
    CONSTRAINT not-null-value CHECK(value NOT NULL));
```

```sql
CREATE TABLE depositor (  
    ....  
    balance  bank-balance,
)
```

Advantages?
Attribute Constraints

Advantage of associating constraints with domains:

1. can avoid repeating specification of same constraint for multiple columns

2. can name constraints
   e.g.: CREATE DOMAIN bank-balance INT (CONSTRAINT not-overdrawn CHECK (value >= 0), CONSTRAINT not-null-value CHECK( value NOT NULL));

allows one to:
1. add or remove:
   ALTER DOMAIN bank-balance
   ADD CONSTRAINT capped
   CHECK( value <= 10000)

2. report better errors (know which constraint violated)
Referential Integrity Constraints

Idea: prevent “dangling tuples” (e.g.: a loan with a bname, *Kenmore*, when no *Kenmore* tuple in branch)

Reference Integrity:
- ensure that:
  - foreign key value $\rightarrow$ primary key value

(note: don’t need to ensure $\leftarrow$, i.e., not all branches have to have loans)
Referential Integrity Constraints

In SQL:

```
CREATE TABLE branch(
    bname CHAR(15) PRIMARY KEY
    ....)

CREATE TABLE loan (
    .......
    FOREIGN KEY bname REFERENCES branch);
```

Affects:
1) Insertions, updates of referencing relation
2) Deletions, updates of referenced relation
what happens when we try to delete this tuple?

Ans: 3 possibilities

1) reject deletion/ update

2) set \( t_i[c], t_j[c] = NULL \)

3) propagate deletion/update
   
   **DELETE:** delete \( t_i, t_j \)
   
   **UPDATE:** set \( t_i[c], t_j[c] \) to updated values
CREATE TABLE A ( ..... 
FOREIGN KEY c REFERENCES B  
............. )

Action: 1) left blank (deletion/update rejected)

2) ON DELETE SET NULL/ ON UPDATE SET NULL
   sets  ti[c] = NULL, tj[c] = NULL

3) ON DELETE CASCADE
   deletes ti, tj
ON UPDATE CASCADE
   sets ti[c], tj[c] to new key values
Global Constraints

Idea: two kinds

1) single relation (constraints spans multiple columns)
   ◦ E.g.: CHECK (total = svngs + check) declared in the CREATE TABLE

2) multiple relations: CREATE ASSERTION

SQL examples:
   1) single relation: All Bkln branches must have assets > 5M

   CREATE TABLE branch (  
       ...........
       bcity CHAR(15),
       assets INT,
       CHECK (NOT(bcity = 'Bkln') OR assets > 5M))

Affects:
   insertions into branch
   updates of bcity or assets in branch
Global Constraints

SQL example:
2) Multiple relations: every loan has a borrower with a savings account

CHECK (NOT EXISTS (  
   SELECT   *  
   FROM loan AS L  
   WHERE NOT EXISTS(  
      SELECT   *  
      FROM borrower B, depositor D, account A  
      WHERE B.cname = D.cname AND  
      D.acct_no = A.acct_no AND  
      L.lno = B.lno))

Problem: Where to put this constraint? At depositor? Loan? ....

Ans: None of the above:  
   CREATE ASSERTION loan-constraint  
   CHECK( ..... )

Checked with EVERY DB update! very expensive.....
## Summary: Integrity Constraints

<table>
<thead>
<tr>
<th>Constraint Type</th>
<th>Where declared</th>
<th>Affects...</th>
<th>Expense</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Constraints</td>
<td>CREATE TABLE (PRIMARY KEY, UNIQUE)</td>
<td>Insertions, Updates</td>
<td>Moderate</td>
</tr>
<tr>
<td>Attribute Constraints</td>
<td>CREATE TABLE CREATE DOMAIN (Not NULL, CHECK)</td>
<td>Insertions, Updates</td>
<td>Cheap</td>
</tr>
<tr>
<td>Referential Integrity</td>
<td>Table Tag (FOREIGN KEY .... REFERENCES ....)</td>
<td>1.Insertions into referencing rel’n</td>
<td>1,2: like key constraints. Another reason to index/sort on the primary keys</td>
</tr>
</tbody>
</table>
|                        |                                                     | 2. Updates of referencing rel’n of relevant attrs | 3,4: depends on
|                        |                                                     | 3. Deletions from referenced rel’n   | a. update/delete policy chosen |
|                        |                                                     | 4. Update of referenced rel’n       | b. existence of indexes on foreign key |
| Global Constraints     | Table Tag (CHECK) or outside table (CREATE ASSERTION) | 1. For single rel’n constraint, with insertion, deletion of relevant attrs | 1. cheap |
|                        |                                                     | 2. For assertions w/ every db modification | 2. very expensive |
Outline

- More SQL
  - Views
  - Integrity Constraints
  - Triggers
  - Transactions
  - Authorization
Authorization and Roles

- Forms of authorization on parts of the database:
  - **Read** - allows reading, but not modification of data.
  - **Insert** - allows insertion of new data, but not modification of existing data.
  - **Update** - allows modification, but not deletion of data.
  - **Delete** - allows deletion of data.

- The `grant` statement is used to confer authorization
  ```
  grant <privilege list>
  on <relation name or view name> to <user list>
  ```

- Examples from PostgreSQL manual:
  - GRANT INSERT ON films TO PUBLIC;
  - GRANT ALL PRIVILEGES ON kinds TO manuel;
Is that it?
  ◦ Unfortunately No
  ◦ SQL 3 standard is several hundreds of pages (if not several thousands)
  ◦ And expensive too..

We will discuss a few more constructs along the way
  E.g. Embedded SQL, creating indexes etc

Again, this is what the reference books are for; you just need to know where to look in the reference book
Outline

- Entity-relationship Model (E/R model)
- Converting from E/R to Relational
- Extra slides
Two key concepts

- **Entities:**
  - An object that *exists* and is *distinguishable* from other objects
    - Examples: Bob Smith, BofA, CMSC424
  - Have *attributes* (people have names and addresses)
  - Form *entity sets* with other entities of the same type that share the same properties
    - Set of all people, set of all classes
  - Entity sets may overlap
    - Customers and Employees
Two key concepts

- **Relationships:**
  - Relate 2 or more entities
    - E.g. Bob Smith *has account at* College Park Branch
  - Form *relationship sets* with other relationships of the same type that share the same properties
    - Customers *have accounts at* Branches
  - Can have attributes:
    - *has account at* may have an attribute *start-date*
  - Can involve more than 2 entities
    - Employee *works at* Branch *at* Job
Entities and relationships

Two Entity Sets

instructor

student

Advisor Relationship, with and without attributes
Alternative representation, used in the book in the past

Both notations used in the rest of the slides
Rest of the class

- Details of the ER Model
  - How to represent various types of constraints/semantic information etc.

- Design issues

- A detailed example
Next: Relationship Cardinalities

- We may know:
  - One customer can only open one account
  - OR
  - One customer can open multiple accounts

- Representing this is important

- Why?
  - Better manipulation of data
    - If former, can store the account info in the customer table
  - Can enforce such a constraint
    - Application logic will have to do it; NOT GOOD
  - Remember: If not represented in conceptual model, the domain knowledge may be lost
Mapping Cardinalities

- Express the number of entities to which another entity can be associated via a relationship set
- Most useful in describing binary relationship sets
Mapping Cardinalities

- One-to-One
- One-to-Many
- Many-to-One
- Many-to-Many
Express the number of entities to which another entity can be associated via a relationship set.

Most useful in describing binary relationship sets.

N-ary relationships?
- More complicated
- Details in the book.

Figure 7.13  E-R diagram with a ternary relationship.
Next: Types of Attributes

- Simple vs Composite
  - Single value per attribute?

- Single-valued vs Multi-valued
  - E.g. Phone numbers are multi-valued

- Derived
  - If date-of-birth is present, age can be derived
  - Can help in avoiding redundancy, enforcing constraints etc...
Types of Attributes

- **Primary key underlined**
  - ID
  - name
  - first_name
  - middle_initial
  - last_name
  - address
  - street
  - street_number
  - street_name
  - apt_number
  - city
  - state
  - zip
  - \{ phone_number \}
  - date_of_birth
  - age ( )

- **Composite**
  - instructor

- **Multi-valued**

- **Derived**
What attributes are needed to represent a relationship completely and uniquely?

- Union of primary keys of the entities involved, and relationship attributes

\{cust-id, access-date, account number\} describes a relationship completely
Is \{\textit{cust-id}, \textit{access-date}, \textit{account number}\} a candidate key?

- No. Attribute \textit{access-date} can be removed from this set without losing key-ness.
- In fact, union of primary keys of associated entities is always a superkey.

![Entity-Relationship Diagram]

- The diagram shows the relationship between a customer and an account.
- The customer entity has a relationship with the account entity through a relationship type called \textit{has}.
- The customer entity has attributes \textit{cust-id} and \textit{access-date}.
- The account entity has an attribute \textit{number}. 

\textit{cust-id} is the primary key associated with the customer entity.
Is \{\text{cust-id, account-number}\} a candidate key?

- Depends
Is \{cust-id, account-number\} a candidate key?

- Depends

- If one-to-one relationship, either \{cust-id\} or \{account-number\} sufficient
  - Since a given customer can only have one account, she can only participate in one relationship
  - Ditto account
Is \{cust-id, account-number\} a candidate key?

- Depends

If one-to-many relationship (as shown), \{account-number\} is a candidate key

- A given customer can have many accounts, but at most one account holder per account allowed
Relationship Set Keys

- General rule for binary relationships
  - one-to-one: primary key of either entity set
  - one-to-many: primary key of the entity set on the many side
  - many-to-many: union of primary keys of the associate entity sets

- n-ary relationships
  - More complicated rules
What have we been doing

Why?

Understanding this is important
  ◦ Rest are details !!
  ◦ That’s what books/manuals are for.
Sometimes a relationship associates an entity set to itself

Need “roles” to distinguish
An entity set without enough attributes to have a primary key

E.g. Section Entity

Still need to be able to distinguish between weak entities

- Called “discriminator attributes”: dashed underline

```
course
- course_id
- title
- credits

sec_course

section
- sec_id
- semester
- year
```
Sometimes a relationship associates an entity set to itself

Need “roles” to distinguish

- **course**: course_id, title, credits
- **sec_course**: sec_id, semester, year
- **section**: sec_id, semester, year
Specialization/Generalization

Similar to object-oriented programming: allows inheritance etc.
Aggregation

- No relationships allowed between relationships
- Suppose we want to record evaluations of a student by a guide on a project
Thoughts...

- Nothing about actual data
  - How is it stored?
- No talk about the query languages
  - How do we access the data?
- Semantic vs Syntactic Data Models
  - Remember: E/R Model is used for conceptual modeling
  - Many conceptual models have the same properties
- They are much more about representing the knowledge than about database storage/querying
Thoughts...

- Basic design principles
  - Faithful
    - Must make sense
  - Satisfies the application requirements
  - Models the requisite domain knowledge
    - If not modeled, lost afterwards
  - Avoid redundancy
    - Potential for inconsistencies
  - Go for simplicity

- Typically an iterative process that goes back and forth
Design Issues

- Entity sets vs attributes
  - Depends on the semantics of the application
  - Consider *telephone*

- Entity sets vs Relationship sets
  - Consider *loan*

- N-ary vs binary relationships
  - Possible to avoid n-ary relationships, but there are some cases where it is advantageous to use them

- It is not an exact science !!
Recap

- **Entity-relationship Model**
  - Intuitive diagram-based representation of domain knowledge, data properties etc...
  - Two key concepts:
    - Entities
    - Relationships
  - We also looked at:
    - Relationship cardinalities
    - Keys
    - Weak entity sets
    - ...
Entity-relationship Model

- No standardized model (as far as I know)
  - You will see different types of symbols/constructs
- Easy to reason about/understand/construct
- Not as easy to implement
  - Came after the relational model, so no real implementation was ever done
  - Mainly used in the design phase
Outline

- Entity-relationship Model (E/R model)
- Converting from E/R to Relational
- Extra slides
Convert entity sets into a relational schema with the same set of attributes

- Customer
  - Cname
  - Ccity
  - Cstreet
  \[ \text{Customer\_Schema}(\text{cname, ccity, cstreet}) \]

- Branch
  - Bname
  - Bcity
  - Assets
  \[ \text{Branch\_Schema}(\text{bname, bcity, assets}) \]
Convert relationship sets \textit{also} into a relational schema

Remember: A relationship is completely described by primary keys of associate entities and its own attributes

Well… Not quite. We can do better. It depends on the relationship cardinality
Say One-to-Many Relationship from Customer to Account → Many accounts per customer

Account_Schema(acct-no, balance, cname, access-date)

Customer_Schema(cname, ccity, cstreet)
# E/R Diagrams → Relations

<table>
<thead>
<tr>
<th>E/R</th>
<th>Relational Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Entity Sets</strong></td>
<td></td>
</tr>
<tr>
<td>$E_i$</td>
<td>$E = (a_1, \ldots, a_n)$</td>
</tr>
<tr>
<td>$a_1 \quad \ldots \quad a_n$</td>
<td></td>
</tr>
</tbody>
</table>

| **Relationship Sets** |  |
| $E_i$ | $R = (a_1, b_1, c_1, \ldots, c_n)$ |
| $E_2$ |  |
| $a_1 \quad \ldots \quad a_n$ | $a_1$: $E_1$’s key |
| $b_1 \quad \ldots \quad b_m$ | $b_1$: $E_2$’s key |
| $c_1 \quad \ldots \quad c_k$ | $c_1, \ldots, c_k$: attributes of R |

*Not the whole story for Relationship Sets ...*
## E/R Diagrams → Relations

<table>
<thead>
<tr>
<th>Relationship Cardinality</th>
<th>Relational Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image.png" alt="Diagram" /></td>
<td></td>
</tr>
</tbody>
</table>
| n:m                       | $E_1 = (a_1, \ldots, a_n)$  
                            | $E_2 = (b_1, \ldots, b_m)$  
                            | $R = (a_1, b_1, c_1, \ldots, c_n)$ |
| n:1                       | $E_1 = (a_1, \ldots, a_n, b_1, c_1, \ldots, c_n)$  
                            | $E_2 = (b_1, \ldots, b_m)$ |
| 1:n                       | $E_1 = (a_1, \ldots, a_n)$  
                            | $E_2 = (b_1, \ldots, b_m, a_1, c_1, \ldots, c_n)$ |
| 1:1                       | Treat as n:1 or 1:n |
Q. How many tables does this get translated into?

A. 6 (account, branch, customer, loan, depositor, borrower)
## E/R Diagrams & Relations

<table>
<thead>
<tr>
<th>E/R</th>
<th>Relational Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Weak Entity Sets</strong></td>
<td></td>
</tr>
<tr>
<td><img src="image1" alt="Diagram" /></td>
<td>Identification Relationship (IR)</td>
</tr>
</tbody>
</table>

**E/R**

- $E_1 = (a_1, \ldots, a_n)$
- $E_2 = (a_1, b_1, \ldots, b_m)$
## E/R Diagrams & Relations

<table>
<thead>
<tr>
<th>E/R</th>
<th>Relational Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Multivalued Attributes</strong></td>
<td></td>
</tr>
<tr>
<td>Employee</td>
<td>Emp = (ssn, name)</td>
</tr>
<tr>
<td></td>
<td>Emp-Phones = (ssn, phone)</td>
</tr>
<tr>
<td>ssn</td>
<td>ssn</td>
</tr>
<tr>
<td>name</td>
<td>name</td>
</tr>
<tr>
<td>phone</td>
<td>phone</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```sql
Emp (ssn, name)
Emp-Phones (ssn, phone)
```
## E/R Diagrams & Relations

<table>
<thead>
<tr>
<th>Subclasses</th>
</tr>
</thead>
</table>

### E/R

<table>
<thead>
<tr>
<th>E/R</th>
<th>Relational Schema</th>
</tr>
</thead>
</table>

### Method 1:

- \( E = (a_1, \ldots, a_n) \)
- \( E_1 = (a_1, b_1, \ldots, b_m) \)
- \( E_2 = (a_1, c_1, \ldots, c_k) \)

### Method 2:

- \( E_1 = (a_1, \ldots, a_n, b_1, \ldots, b_m) \)
- \( E_2 = (a_1, \ldots, a_n, c_1, \ldots, c_k) \)
Subclasses example:

Method 1:

- Account = (acct\_no, balance)
- SAccount = (acct\_no, interest)
- CAccount = (acct\_no, overdraft)

Method 2:

- SAccount = (acct\_no, balance, interest)
- CAccount = (acct\_no, balance, overdraft)