The Relational Model

COMP 3380 - Databases: Concepts and Usage

Department of Computer Science
The University of Manitoba
Fall 2006

Review: Conceptual/Logical Data Model

- Examples:
  1. **Entity-Relationship model**
     - **Entities** are relations (i.e., tables with rows & column)
     - **Relationships** describe association between entities
  2. **Relational model - *most widely used model***
     - **Relations** are basically tables with rows & columns

Why Study the Relational Model?

- Proposed by E. F. Codd (1923-2003) in 1970
- Most widely used model
  - Vendors: IBM (DB2), Informix, Microsoft (Access), Oracle, Sybase, etc.
- "Legacy systems" in older models
  - E.g., IBM's IMS
- Recent competitor: Object-oriented model
  - E.g., ObjectStore, Versant, Ontos
  - A synthesis emerging: **Object-relational model** (e.g., Informix Universal Server, UniSQL, O2, Oracle, DB2)

Relational DB

- **Relational database***:* A set of *relations*
- **Relation:** Made up of 2 parts
  1. **Instance:** A *table*, with rows & columns
     - #rows = cardinality, #columns = degree *or* arity
  2. **Schema:** Basic info describing a table or relation (i.e., specifies name of relation, plus name & type of each column)
     - E.g., *Students (sID: string, sName: string, loginID: string)*
- Can think of a relation instance as *a* ***set*** *of rows* (i.e., **all rows/ tuples/ records are distinct**)

Example: Instance of Students Relation

Students

sID	sName	loginID
3666	Adam	adam@cs
3688	Ben	ben@ece

- Cardinality (i.e., #rows *or* #tuples) = 2
- Degree (aka arity; i.e., #cols *or* #attrs *or* #fields) = 3
- All rows are distinct
- **Order of rows is unimportant**

Relational DB: Relation Instance

- Given *n* domain sets D_1, D_2, …. D_n, a **relation instance** *r* is
  - a subset of $D_1 \times D_2 \times \ldots \times D_n$
  - a set of n-tuples (a_1, a_2, …, a_n) where each $a_i \in D_i$
- A domain D_i
  - is the set of allowed values for each attribute a_i (normally required to be *atomic,* i.e., not multi-valued & not composite)
  - contains a special value **NULL** indicating that the value is not known (or is non-applicable)

Example: Relation Instance

- If *custName* = {Adam, Ben, Charles, Don},
  custStreet = {Robson, St-Catherine, Yonge}, &
  custCity = {Montréal, Toronto, Vancouver},
  then, *Customer r* = { (Adam, St-Catherine, Montréal),
  (Ben, Yonge, Toronto),
  (Charles, Yonge, Toronto),
  (Don, Robson, Vancouver) } is a relation instance over *custName* x *custStreet* x *custCity*

Relational DB: Relation Schema

- If A_1,…, A_n are attributes with domains D_1,..., D_n, then R (A_1:D_1, …, A_n :D_n) is a **relation schema** that defines a relation type
- E.g., *Customer (custName: string,*
  custStreet: string,
  custCity: string)
- **Normalization theory** deals with how to design relation schemas
- A DB consists of multiple relations

Creating Relations

- Data definition language (DDL), such as SQL, for creating Students relation

```
CREATE TABLE Students
      (sID CHAR(20),
       sName CHAR(20),
       loginID CHAR(10))
```

- Creates Enrolled relation

```
CREATE TABLE Enrolled
      (sID CHAR(20),
       cID CHAR(20),
       grade CHAR(2))
```


Destroying and Altering Relations

- Destroys the relation Students. The schema information *and* the tuples are deleted.

```
DROP TABLE Students
```

- Alters the schema of Students by adding a new column (called firstYear); every tuple in the current instance is extended with a NULL value in the new column.

```
ALTER TABLE Students
      ADD firstYear INTEGER
```

Adding Tuples

- Data manipulation language (DML), such as SQL, for inserting a single tuple

INSERT INTO Students (sID, sName, loginID)
VALUES ('3666', 'Adam', 'adam@cs')

or

INSERT INTO Students
VALUES ('3666', 'Adam', 'adam@cs')

Adding Tuples

- Inserts tuples based on the result of a query

INSERT INTO Freshmen (sID, sName, loginID)
SELECT sID, sName, loginID
FROM Students
WHERE firstYear = 1

Deleting and Modifying Tuples

- Deletes all tuples satisfying some conditions (e.g., name = 'Adam')

```
DELETE FROM Students S
    WHERE S.sName = 'Adam'
```

- Changes attributes (e.g., grade) of all tuples satisfying some conditions (e.g., sID='3666' & cID='comp3380')

```
UPDATE Enrolled E
    SET E.grade = 'A'
    WHERE E.sID = '3666' and E.cID = 'comp3380'
```


Integrity Constraints (ICs)

- **Integrity constraints (ICs)** are conditions that must be true for *any* instance of the database (e.g., domain constraints)
  - **ICs are specified when schema is defined**
  - **ICs are checked when relations are modified** (e.g., insertions/ deletions/ updates that violate ICs are disallowed)
- A **legal instance** of a relation is one that satisfies all specified ICs
  - DBMS should not allow illegal instances

Integrity Constraints (ICs)

- ICs can be used to ensure application semantics (e.g., *sID* is a key), or to prevent inconsistencies (e.g., *sName* has to be a string, *age* must be < 150)
- If the DBMS checks ICs, stored data is more faithful to real-world meaning
  - Avoids data entry errors, too!

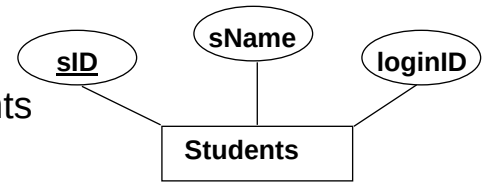
Types of Integrity Constraints (ICs)

- **Domain constraints**
  - Attribute values must be of right type. Always enforced.
- **Key constraints** (aka **primary key constraints**)
  - Forbid duplicate primary key values (i.e., no 2 tuples have the same primary key value; *primary key value is unique*)
- **Entity integrity constraints**
  - **Primary key is NOT NULL**
- **Referential integrity constraints** (aka **foreign key constraints**)
  - Ensure consistency of tuple references across tables (i.e., no dangling references)
- Other general constraints (e.g., semantics)

Possible Violations of Integrity Constraints (ICs)

- Recall: ICs are specified when schema is defined; ICs are checked when relations are modified (e.g., insertions, deletions, updates)
- Insertions
  - may violate (1) domain constraints, (2) key constraints (primary key constraints), (3) entity integrity constraints, & (4) referential integrity constraints (foreign key constraints)
- Deletions
  - may violate (4) referential integrity constraints
- Updates (~ deletions follow by insertions)
  - may violate (1) domain constraints, (2) key constraints, (3) entity integrity constraints, & (4) referential integrity constraints

Review: Primary Key Constraints



- **Superkey**
  - A set of one or more attributes that (taken collectively) identify uniquely an entity in an entity set
  - E.g., {sID}, {sID, sName}, {sID, sName, loginID}, {sID, loginID}, {loginID}, {loginID, sName}
- **Candidate key**
  - Minimal superkey (i.e., a superkey for which no proper subset is a superkey)
  - E.g., {sID}, {loginID} **More than 1 candidate keys!**
- **Primary key**
  - The candidate key chosen as the principal means to identify entities in an entity set
  - E.g., {sID}

Primary and Candidate Keys

- Possibly many **candidate keys** (specified using **UNIQUE**), one of which is chosen as the **primary key**.

```
CREATE TABLE Students
      (sID CHAR(20) NOT NULL,        ← For some DBMS, this
       sName CHAR(20),                  "NOT NULL" is optional
       loginID CHAR(10) NOT NULL,
       PRIMARY KEY (sID),
       UNIQUE (loginID))
```


Foreign Keys & Referential Integrity

- **Foreign key:** Set of attributes in one relation (i.e., ***referencing relation***) that is used to "refer" to a tuple in another relation (i.e., ***referenced relation***).
  - Must correspond to **candidate key of the referenced relation**. Like a "logical pointer".
- E.g., sID is a foreign key referring to Students
  - *Enrolled (sID: string, cID: string, grade: string)*
  - If all foreign key constraints are enforced, **referential integrity** is achieved (i.e., no dangling references).
  - Can you name a data model without referential integrity?
    - Links in HTML!

Foreign Keys

- Only students listed in the Students relation should be allowed to enrol for courses.

```
CREATE TABLE Enrolled
      (sID CHAR(20) NOT NULL,
       cID CHAR(20) NOT NULL,
       grade CHAR(2),
       PRIMARY KEY (sID, cID),
       FOREIGN KEY (sID) REFERENCES Students,
       FOREIGN KEY (cID) REFERENCES Courses)
```


Enforcing Referential Integrity

- Consider Students and Enrolled; *sID* in Enrolled is a foreign key that references Students.
- What should be done if an Enrolled tuple with a non-existent student ID is inserted?
  - Reject it!

Enforcing Referential Integrity

- What should be done if a Students tuple is deleted?
  - **Default:** Disallow deletion of a Students tuple that is referred to.
  - **Other Options:**
    - Also delete all Enrolled tuples that refer to it (via the use of **ON DELETE CASCADE**).
    - Set *sID* in Enrolled tuples that refer to it to a *default sID* (via the use of **ON DELETE SET DEFAULT**).
    - In SQL: Set *sID* in Enrolled tuples that refer to it to a special value NULL (via the use of **ON DELETE SET NULL**).
- Similar if primary key of Students tuple is updated.

Examples 1 & 2: Referential Integrity

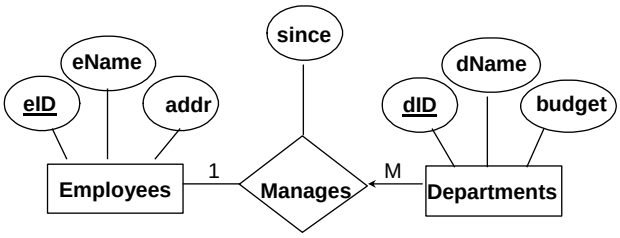
- Default (i.e., ON DELETE NO ACTION):

```
CREATE TABLE Enrolled
      (sID CHAR(20) NOT NULL,
       cID CHAR(20) NOT NULL,
       grade CHAR(2),
       PRIMARY KEY (sID, cID),
       FOREIGN KEY (sID) REFERENCES Students,
       FOREIGN KEY (cID) REFERENCES Courses)
```

- Option 1 - ON DELETE CASCADE:

```
... FOREIGN KEY (sID) REFERENCES Students
          ON DELETE CASCADE, …
```

Example 3: Referential Integrity



- Option 2 - ON DELETE SET DEFAULT:

CREATE TABLE Manages (
dID INTEGER **NOT NULL**,
eID INTEGER ***DEFAULT*** 0,
since DATE,
PRIMARY KEY (dID),
FOREIGN KEY (eID) **REFERENCES** Employees
ON DELETE SET DEFAULT,
FOREIGN KEY (dID) **REFERENCES** Departments)

Example 4: Referential Integrity

- Option 3 - ON DELETE SET NULL:

CREATE TABLE Manages (
dID INTEGER **NOT NULL**,
eID INTEGER,
since DATE,
PRIMARY KEY (dID),
FOREIGN KEY (eID) **REFERENCES** Employees
ON DELETE SET NULL,
FOREIGN KEY (dID) **REFERENCES** Departments)

Where Do ICs Come From?

- ICs are based upon the semantics of the real-world enterprise that is being described in the database relations
- By looking at an instance:
  - We **can check a database instance to see if an IC is violated**
  - We **can NEVER infer that an IC is true**
  - **An IC is a statement about *all possible* instances!**
  - From the example, we know *sName* is not a key, but the assertion that *sID* is a key is given to us.
- Key and foreign key ICs are the most common; more general ICs supported too.

Views

- A **view** is just a relation, but we store a *definition* (rather than a set of tuples).

```
CREATE VIEW YoungActiveStudents (sID, sName, grade) AS
    SELECT S.sID, S.sName, E.grade
    FROM Students S, Enrolled E
    WHERE S.sID = E.sID and S.age < 21
```

- Views can be dropped using the **DROP VIEW** command.

```
DROP VIEW YoungActiveStudents
```

Views and Security

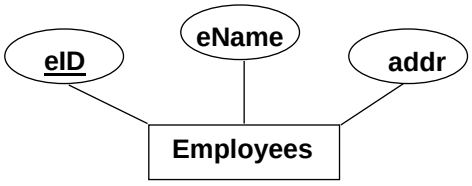
- Views provides the support for *logical data independence*
- Views can be used to present necessary information (or a summary), while *hiding details* in underlying relation(s).
  - Given YoungActiveStudents (but not Students or Enrolled), we can find students who are enrolled in some courses without revealing the *cID*s of the courses they are enrolled in.

Logical DB Design: ER to Relational Model

Review: Database Design Process

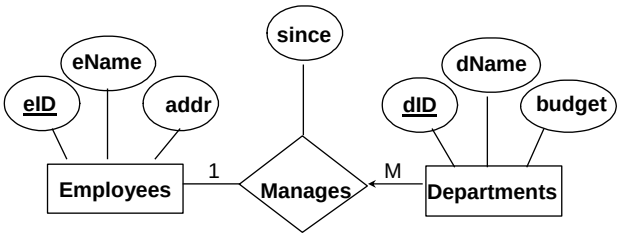
1. Requirements collection & analysis
2. Conceptual DB design (ER model)
3. **Logical design (data model mapping, e.g., map ER to tables)**
4. Schema refinement (e.g., normalization)
5. Physical design
6. System implementation & tuning (e.g., application & security design)

Logical DB Design: ER to Relational Model



CREATE TABLE Employees
(eID INTEGER **NOT NULL**,
eName CHAR(20),
addr CHAR(100),
PRIMARY KEY (eID))

Logical DB Design: ER to Relational Model

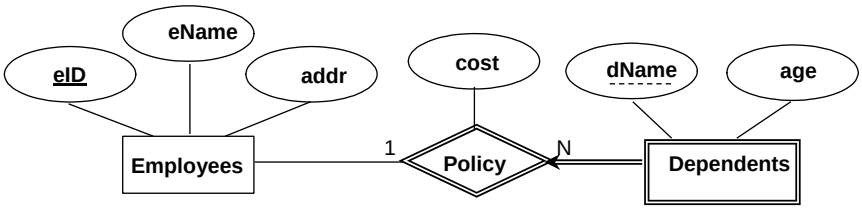


CREATE TABLE Manages (
dID INTEGER **NOT NULL**,
eID INTEGER,
since DATE,
PRIMARY KEY (dID),
FOREIGN KEY (eID) **REFERENCES** Employees,
FOREIGN KEY (dID) **REFERENCES** Departments)

Logical DB Design: ER to Relational Model

- **One table for each entity set**
  - Attrs are the attrs of the entity set
- **One table for each relationship set**
  - Attrs include the primary keys of participating entity sets (as foreign keys) & the descriptive attrs of the relationship set
- Alternative:
  - Include the info about a relationship set (e.g., *Manages*) in the table corresponding to the relevant entity set (e.g., *Dept*)
  - E.g., merge *Dept (dID, dName, budget)*
    and *Manages (dID, eID, since)*
    into *DeptMgr (dID, dName, budget, eID, since)*

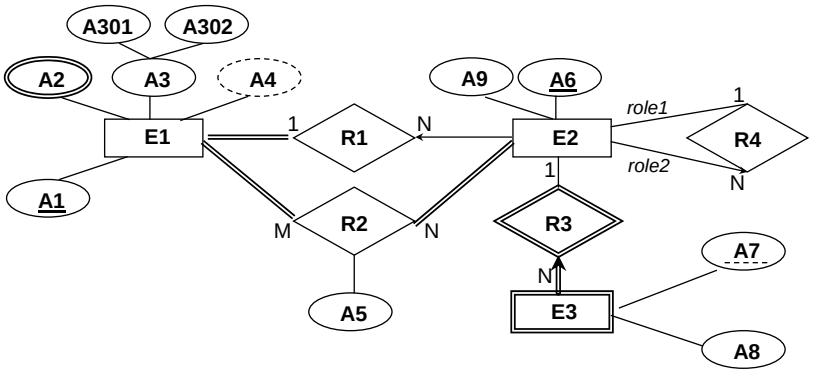
Logical DB Design: ER to Relational Model



CREATE TABLE DependentPolicy (
eID CHAR(11) **NOT NULL**, dName CHAR(20) **NOT NULL**,
age INTEGER, cost REAL,
PRIMARY KEY (eID, dName),
FOREIGN KEY (eID) **REFERENCES** Employees
ON DELETE CASCADE)

Logical DB Design: ER to Relational Model

- ER Model:



Logical DB Design: ER to Relational Model

- Relational Model:
  WLOG, let us assume that all attributes are of type CHAR(20)

- **CREATE TABLE** E1 (
  A1 CHAR(20) **NOT NULL**,
  A301 CHAR(20),
  A302 CHAR(20),
  PRIMARY KEY (A1))

- **CREATE TABLE** E2 (
  A6 CHAR(20) **NOT NULL**,
  A9 CHAR(20),
  PRIMARY KEY (A6))

Logical DB Design: ER to Relational Model

- **CREATE TABLE** mvA2 (
  A1 CHAR(20) **NOT NULL**,
  a2 CHAR(20) **NOT NULL**,
  PRIMARY KEY (A1, a2),
  FOREIGN KEY (A1) **REFERENCES** E1)

- Note:
  a2 (e.g., author) is a single-valued attribute for storing each of the multiple A2 values (e.g., authors)

Logical DB Design: ER to Relational Model

- **CREATE TABLE** E3 (
  A6 CHAR(20) **NOT NULL**,
  A7 CHAR(20) **NOT NULL**,
  A8 CHAR(20),
  PRIMARY KEY (A6, A7),
  FOREIGN KEY (A6) **REFERENCES** E2
  ON DELETE CASCADE)

Logical DB Design: ER to Relational Model

- **CREATE TABLE** R1 (
  A1 CHAR(20),
  A6 CHAR(20) **NOT NULL**,
  PRIMARY KEY (A6),
  FOREIGN KEY (A1) **REFERENCES** E1,
  FOREIGN KEY (A6) **REFERENCES** E2)
- **CREATE TABLE** R2 (
  A1 CHAR(20) **NOT NULL**,
  A6 CHAR(20) **NOT NULL**,
  A5 CHAR(20),
  PRIMARY KEY (A1,A6),
  FOREIGN KEY (A1) **REFERENCES** E1,
  FOREIGN KEY (A6) **REFERENCES** E2)

Logical DB Design: ER to Relational Model

- **CREATE TABLE** R4 (
  $A6_1$ CHAR(20),
  $A6_2$ CHAR(20) **NOT NULL**,
  PRIMARY KEY ($A6_2$),
  FOREIGN KEY ($A6_1$) **REFERENCES** E2(A6),
  FOREIGN KEY ($A6_2$) **REFERENCES** E2(A6))

Logical DB Design: ER to Relational Model

- Relational Model:
  - E1 (A1, A301, A302)
  - E2 (A6, A9)
  - mvA2 (A1, a2) *where* foreign key (A1) references E1
  - E3 (A6, A7, A8) *where* foreign key (A6) references E2
    on delete cascade
  - R1 (A1, A6) *where* foreign key (A1) references E1,
    foreign key (A6) references E2
  - R2 (A1, A6, A5) *where* foreign key (A1) references E1,
    foreign key (A6) references E2
  - R4 ($A6_1$, $A6_2$) *where* foreign key ($A6_1$) references E2(A6),
    foreign key ($A6_2$) references E2(A6)

Logical DB Design: ER to Relational Model

- Alternative:

  Similar to the *Dept/Manages/DeptMgr* example, E2 can be merged with R1 and R4:
  - E2 (A6, A9)
  - R1 (A1, A6) *where* foreign key (A1) references E1,
    foreign key (A6) references E2
  - R4 ($A6_1$, $A6_2$) *where* foreign key ($A6_1$) references E2(A6),
    foreign key ($A6_2$) references E2(A6)

  to form E2':
  - E2' (A6, A9, A1, $A6_1$) *where* foreign key (A1) references E1,
    foreign key ($A6_1$) references E2'(A6)

Logical DB Design: ER to Relational Model

- Relational Model (after the merge):
  - E1 (A1, A301, A302)
  - mvA2 (A1, a2) *where* foreign key (A1) references E1
  - E2 (A6, A9, A1, $A6_1$) *where* foreign key (A1) references E1,
    foreign key ($A6_1$) references E2(A6)
  - E3 (A6, A7, A8) *where* foreign key (A6) references E2
    on delete cascade
  - R2 (A1, A6, A5) *where* foreign key (A1) references E1,
    foreign key (A6) references E2

Comments on Logical DB Design: Textbook Approach

1. Mapping of strong entity sets
2. Mapping of weak entity sets
3. Mapping of binary 1-to-1 relationship sets
4. Mapping of binary 1-to-many relationship sets
5. Mapping of binary many-to-many relationship sets
6. Mapping of multi-valued attributes
7. Mapping of *n*-ary relationship sets

➔ *Gives the same relational model!*

Comments on Logical DB Design: Textbook Approach

Example:

1. **E1 (A1, A301, A302)**
   E2 (A6, A9)
2. **E3 (A6, A7, A8) *where* foreign key (A6) references E2 on delete cascade**
4. With R1 (A1, A6), E2 becomes:
   E2 (A6, A9, A1) *where* foreign key (A1) references E1
   With R4 ($A6_1$, $A6_2$), E2 then becomes:
   E2 (A6, A9, A1, $A6_1$) *where* foreign key (A1) references E1, foreign key ($A6_1$) references E2(A6)
5. **R2 (A1, A6, A5) *where* foreign key (A1) references E1, foreign key (A6) references E2**
6. **mvA2 (A1, a2) *where* foreign key (A1) references E1**

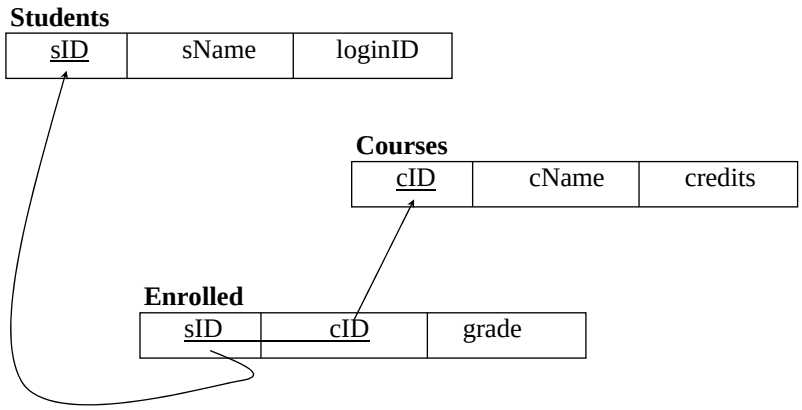
Comments on Logical DB Design: Notation 1

- **CREATE TABLE** Students
  (sID CHAR(20) **NOT NULL,** sName CHAR(20),
  loginID CHAR(10), **PRIMARY KEY** (sID))
- **CREATE TABLE** Courses
  (cID CHAR(20) **NOT NULL,** cName CHAR(20),
  credits INTEGER, **PRIMARY KEY** (cID))
- **CREATE TABLE** Enrolled
  (sID CHAR(20), cID CHAR(20),
  grade CHAR(2)) **PRIMARY KEY** (sID, cID),
  FOREIGN KEY (sID) **REFERENCES** Students,
  FOREIGN KEY (cID) **REFERENCES** Courses)

Comments on Logical DB Design: Notation 2

- Students (<u>sID</u>, sName, loginID)
- Courses (<u>cID</u>, cName, credits)
- Enrolled (<u>sID, cID</u>, grade) *where*
  foreign key (sID) references Students,
  foreign key (cID) references Courses

Comments on Logical DB Design: Notation 3



Summary of Relational Model

- A tabular representation of data
- Simple & intuitive, currently the most widely used
- Integrity constraints can be specified by the DBA, based on application semantics. DBMS checks for violations.
  - Two important ICs: Primary and foreign keys
  - In addition, we *always* have domain constraints
- Powerful & natural query languages exist

Summary of Relational Model

- Rules to translate ER to relational model:
  - One table for each entity set
  - One table for each multi-valued attribute of the entity set
  - One table for each relationship set
    - Except for the identifying relationship set of the weak entity set
  - Tables could be merged (if necessary)