Translating Entity-Relationship to Relational Tables

Spring 2018

School of Computer Science
University of Waterloo

Databases CS348
E-R Diagram to Relational Schema

Main ideas:
- Each entity set maps to a new table
- Each attribute maps to a new table column
- Each relationship set maps to either new table columns or to a new table
Representing Strong Entity Sets

Entity set $E$ with attributes $a_1, \ldots, a_n$ translates to table $E$ with attributes $a_1, \ldots, a_n$

Entity of type $E \leftrightarrow$ row in table $E$

Primary key of entity set $\rightarrow$ primary key of table

**Example:**

Student

<table>
<thead>
<tr>
<th>StudentNum</th>
<th>StudentName</th>
<th>Major</th>
</tr>
</thead>
</table>

(University of Waterloo)
Representing Weak Entity Sets

Weak entity set $E$ translates to table $E$

Columns of table $E$ should include

- Attributes of the weak entity set
- Attributes of the identifying relationship set
- Primary key attributes of entity set for dominating entities

Primary key of weak entity set $\rightarrow$ primary key of table
Representing Weak Entity Sets (cont.)

Example:

ER to Relational
Representing Relationship Sets

- If the relationship set is an identifying relationship set for a weak entity set then no action needed.

- If we can deduce the general cardinality constraint (1,1) for a component entity set \( E \) then add following columns to table \( E \):
  - Attributes of the relationship set
  - Primary key attributes of remaining component entity sets

- Otherwise: relationship set \( R \rightarrow \) table \( R \)
Representing Relationship Sets (cont.)

- Columns of table \( R \) should include
  - Attributes of the relationship set
  - Primary key attributes of each component entity set

- Primary key of table \( R \) determined as follows
  - If we can deduce the general cardinality constraint \((0,1)\) for a component entity set \( E \), then take the primary key attributes for \( E \)
  - Otherwise, choose primary key attributes of each component entity
Representing Relationship Sets (cont.)

Example:

Note that the role name of a component entity set should be prepended to its primary key attributes, if supplied.
Representing Aggregation

Tabular representation of aggregation of $R$

$= \text{tabular representation for relationship set } R$

To represent relationship set involving aggregation of $R$, treat the aggregation like an entity set whose primary key is the **primary key** of the table for $R$.
Representing Aggregation (cont.)

Example:

```
<table>
<thead>
<tr>
<th>StudentNum</th>
<th>CourseNum</th>
<th>ExpirationDate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Account</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Course</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Student</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EnrolledIn</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>StudentNum</th>
<th>CourseNum</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Account</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Course</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Representing Specialization

Create table for higher-level entity set, and treat specialized entity subsets like weak entity sets (without discriminators)

Example:

```
Student
  StudentNumber
  StudentName

Graduate
  StudentNumber
  ProfessorName

Degrees
  StudentNumber

SupervisedBy
  (1, 1)

Professor
  ProfessorName
```

(University of Waterloo)
Representing Generalization (Approach #1)

Create a table for each lower-level entity set only

Columns of new tables should include
- Attributes of lower level entity set
- Attributes of the superset

The higher-level entity set can be defined as a view on the tables for the lower-level entity sets
Representing Generalization (Approach #1)

Example:

```
<table>
<thead>
<tr>
<th>LicenceNum</th>
<th>MakeAndModel</th>
<th>Price</th>
<th>Tonnage</th>
<th>AxelCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Truck</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LicenceNum</td>
<td>MakeAndModel</td>
<td>Price</td>
<td>MaxSpeed</td>
<td>PassengerCount</td>
</tr>
<tr>
<td>Car</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Representing Generalization (Approach #2)

Treat generalization the same as specialization.

Example:

```
Vehicle
  LicenceNum  MakeAndModel  Price
Truck
  LicenceNum  Tonnage  AxelCount
Car
  LicenceNum  MaxSpeed  PassengerCount
```

```
MakeAndModel
LicenceNum
Vehicle
Price

Tonnage
Truck
AxelCount

Car
MaxSpeed
PassengerCount

COVERS
LicenceNum  MaxSpeed  PassengerCount
Car
LicenceNum  MakeAndModel  Price
Vehicle
LicenceNum  Tonnage  AxelCount
Truck
```

(University of Waterloo)
Example Translation: ER Diagram

- **CourseNum**
- **Course**
  - **CourseName**
  - **Term**
  - **TaughtBy**
    - **Professor**
      - **ProfName**
      - **ProfNum**
    - **EnrolledIn**
      - **Student**
        - **StudentName**
        - **StudentNum**
        - **Location**
        - **Mark**
        - **GPA**
      - **Section**
        - **Course**
          - **CourseName**
        - **SectionOf**
          - **Term**
        - **EnrolledIn**
          - **Student**
          - **Location**
          - **Mark**
          - **GPA**

- **(0, N)**
- **(1, 1)**
- **(6, 50)**
Example Translation: Relational Diagram

Course
- CourseNum
- CourseName

Section
- CourseNum
- SectionNum
- Term
- ProfNum

Off-Site Section
- CourseNum
- SectionNum
- Term
- Location

Student
- StudentNum
- StudentName
- GPA

EnrolledIn
- CourseNum
- SectionNum
- StudentNum
- Term
- Mark

Professor
- ProfNum
- ProfName
Defining Relations and Integrity Constraints in SQL

- connected with a table (definition)
  - Primary Keys
  - Foreign Keys
  - CHECK constraints

- separate ECA rules (triggers)
Tables with a Primary Key

- specifies a *attributes* of a table (w/types)
- specifies a *primary key* in a table
- syntax:
  
  ```
  CREATE TABLE <name> 
  (  ... <attributes>,
  PRIMARY KEY ( <list of attr> ) 
  )
  ```

- also creates an unique index on the key
Example

```
create table DEPT
  ( ID integer not NULL,
    DeptName char(20),
    MgrNO char(3),
    PRIMARY KEY (ID)
  )
```
Example (cont.)

sql => insert into DEPT values \
sql (cont.) => ( 1 , 'Computer Science', 000100)
DB20000I  The SQL command completed successfully.

sql => insert into DEPT values \
sql (cont.) => ( 1 , 'Computer Science', 000100)
SQL0803N  One or more values in the INSERT or UPDATE statement are not valid because they would produce duplicate rows for a table with a unique index.
SQLSTATE=23505

(University of Waterloo)
Foreign Key

- specifies an *referential constraint*

- syntax:

  ```sql
  CREATE TABLE <name>
  (   ... <attributes>,
      FOREIGN KEY ( <attrs> )
      REFERENCES <ref-table>( <attrs> )
      ON DELETE <delete-action>
      ON UPDATE <update action>
  )
  ```

- the actions can be:

  - *RESTRICT* – produce an error
  - *CASCADE* – propagate the delete
  - *SET NULL* – set to “unknown”
create table EMP
  ( SSN integer not NULL,
    Name char(20),
    Dept integer,
    Salary dec(8,2),
    primary key (SSN),
    foreign key (Dept) references DEPT(ID)
    on delete cascade
    on update restrict)
Example (cont.)

db2 => insert into EMP \\nsql (cont.) => values ( 999, 'DAVE', 2, 50000 )
SQL0530N The insert or update value of FOREIGN KEY "DAVID.EMP.SQL970916001756640" is not equal to any value of the primary key of the parent table. SQLSTATE=23503

db2 => insert into EMP \\nsql (cont.) => values ( 999, 'DAVE', 1, 50000 )
DB20000I The SQL command completed successfully.

db2 => select * from emp where SSN=999

<table>
<thead>
<tr>
<th>SSN</th>
<th>NAME</th>
<th>DEPT</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>999</td>
<td>DAVE</td>
<td>1</td>
<td>50000.00</td>
</tr>
</tbody>
</table>
Example (cont.)

db2 => delete from DEPT where id=1
DB20000I The SQL command completed successfully.
db2 => select * from emp where SSN=999

<table>
<thead>
<tr>
<th>SSN</th>
<th>NAME</th>
<th>DEPT</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CHECK constraints

- allow checking for “correct” data:
- syntax:

  ```sql
  CREATE TABLE <name>
  (    ... <attributes>,
      CHECK <condition>
  )
  ```
- condition is a *simple* search condition
  ⇒ no subqueries (in DB2)
Example

create table EMP
  ( SSN integer not NULL,
    Name char(20),
    Dept integer,
    Salary dec(8,2),
    primary key (SSN),
    foreign key (Dept) references DEPT(ID)
      on delete cascade
      on update restrict,
    check ( salary > 0 )
  )

db2 => insert into emp values (998, 'DAVE', 1, 0 )
SQL0545N  The requested operation is not allowed
because a row does not satisfy the check constraint
"DAVID.EMP.SQL970916000939620".  SQLSTATE=23513
Views

Definition (View)

A view is a relation whose instance is determined by the instances of other relations.

A view has many of the same properties as a table.

- its schema information appears in the database schema
- access controls can be applied to it
- other views can be defined in terms of it
Types of Views

- **Virtual**: Views are used only for querying; they are not stored in the database.
- **Materialized**: The query that makes up the view is executed, the view constructed and stored in the database.
SQL DDL: Views

- **General form:**
  
  ```sql
  CREATE VIEW <name>
  [AS] ( <query> )
  ```

- **Example**
  
  ```sql
  create view ManufacturingProjects
  ( Select projno, projname, firstnme, lastname
  From project, employee
  Where respemp = empno and deptno = 'D21' )
  ```
Accessing a View

Query a view as if it were a base relation.

SELECT projname
FROM manufacturingprojects

What happens when you query a virtual view?

- At compile time, the view definition is found
- The query over the view is modified with the query definition
- The resulting query is optimized and executed
Updating Views

- Modifications to a view’s instance must be propagated back to instances of relations in conceptual schema.
- Some views cannot be updated unambiguously.

Conceptual Schema

<table>
<thead>
<tr>
<th>Persons</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>CITIZENSHIP</td>
</tr>
<tr>
<td>Ed</td>
<td>Canadian</td>
</tr>
<tr>
<td>Dave</td>
<td>Canadian</td>
</tr>
<tr>
<td>Wes</td>
<td>American</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NationalPastimes</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CITIZENSHIP</td>
<td>PASTIME</td>
</tr>
<tr>
<td>Canadian</td>
<td>Hockey</td>
</tr>
<tr>
<td>Canadian</td>
<td>Curling</td>
</tr>
<tr>
<td>American</td>
<td>Hockey</td>
</tr>
<tr>
<td>American</td>
<td>Baseball</td>
</tr>
</tbody>
</table>

External Schema

<table>
<thead>
<tr>
<th>PersonalPastimes</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>PASTIME</td>
</tr>
<tr>
<td>Ed</td>
<td>Hockey</td>
</tr>
<tr>
<td>Ed</td>
<td>Curling</td>
</tr>
<tr>
<td>Dave</td>
<td>Hockey</td>
</tr>
<tr>
<td>Dave</td>
<td>Curling</td>
</tr>
<tr>
<td>Wes</td>
<td>Hockey</td>
</tr>
<tr>
<td>Wes</td>
<td>Baseball</td>
</tr>
</tbody>
</table>

1. What does it mean to insert (Darryl, Hockey)?
2. What does it mean to delete (Dave, Curling)?
According to SQL-92, a view is updatable only if its definition satisfies a variety of conditions:

- The query references exactly one table
- The query only outputs simple attributes (no expressions)
- There is no grouping/aggregation/$DISTINCT$
- There are no nested queries
- There are no set operations

These rules are more restrictive than necessary.
Materialized Views

Problem
When a base table changes, the materialized view may also change.

Solution?
- Periodically reconstruct the materialized view.
- Incrementally update the materialized view.

Example: Data warehouses
Data Control Language

assigns *access rights* to database objects

**Syntax:**

- `GRANT <what> ON <object> TO <user(s)>`
- `REVOKE <what> ON <object> FROM <user(s)>`

**<what> ON <object> can be**

- **DATABASE:** BINDADD, CONNECT, CREATETAB, CREATE_NOT_FENCED, DBADM
- **INDEX:** CONTROL
- **PACKAGE:** BIND, CONTROL, EXECUTE
- **TABLE/VIEW:** ALTER, CONTROL, INDEX, REFERENCES, SELECT, INSERT, DELETE, UPDATE

**<user(s)> is a list of**

(i) USER <name>  (ii) GROUP <name>  (iii) PUBLIC
Summary

Schema design summary:

1. Create an ER diagram
   ⇒ visualization of the design goals
2. Translate ER-to-Relational
3. Determine FD, MVD, JD, . . .
   ⇒ detect anomalies and decompose
   ⇒ find keys
4. Determine inter-relational constraints
   ⇒ INDs and foreign key constraints
5. Enforce rest of constraints
   ⇒ CHECK declarations
   ⇒ ECA rules (only as the last resort!)