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 Num  StudentName  Major
```

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:

![Entity Relationship Diagram]

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 \(R \rightarrow \text{table } R\)

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

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 $R$.

Example:

\[
\begin{array}{cccc}
\text{Student} & \text{Course} & \text{Account} & \text{UserId} \\
\text{StudentNum} & \text{CourseNum} & \text{ExpirationDate} \\
\end{array}
\]

Representing Specialization

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

Example:

\[
\begin{array}{cccc}
\text{Student} & \text{Graduate} & \text{Degrees} & \text{SupervisedBy} \\
\text{StudentNumber} & \text{StudentName} & \text{ProfessorName} \\
\text{StudentNumber} & \text{ProfessorName} & \text{Degree} \\
\end{array}
\]

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:

![Diagram of ER to Relational transformation for Approach #1]

Representing Generalization (Approach #2)

Treat generalization the same as specialization.

Example:

![Diagram of ER to Relational transformation for Approach #2]

Example Translation: ER Diagram

![Diagram of ER diagram for Example Translation]

Example Translation: Relational Diagram

![Diagram of Relational diagram for Example Translation]
Defining Relations and Integrity Constraints in SQL

- connected with a table (definition)
  - Primary Keys
  - Foreign Keys
  - CHECK constraints
- separate ECA rules (triggers)

**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', 001000)
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

Tables with a Primary Key

- specifies a attributes of a table (w/types)
- specifies a primary key in a table
- syntax:

```sql
CREATE TABLE <name>
  (   ... <attributes>,
      PRIMARY KEY ( <list of attr> )
  )
```
- also creates an unique index on the key
Foreign Key

- specifies an *referential constraint*
- syntax:

  ```
  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”

Example

```sql
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 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**

```sql
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:
  
  CREATE VIEW <name>
  [AS] ( <query> )

- Example
  
  create view ManufacturingProjects
  ( Select projno, projname, firstname, 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.

View Updates in SQL

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/VIEWS: 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!)