Relational Model

• A relational database consists of a group of relations (a.k.a., tables)
• A relation (table) is a set of tuples (rows)
• The tuples (rows) have a fixed set of fields (columns)
ER Entity Sets

• An entity set in the ER model maps to a table in the relational model
  – The entity set's attributes map to columns

<table>
<thead>
<tr>
<th>employees</th>
<th>ssn</th>
<th>last_name</th>
<th>first_name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>111-11-1111</td>
<td>Bunny</td>
<td>Bugs</td>
</tr>
<tr>
<td></td>
<td>333-33-3333</td>
<td>Duck</td>
<td>Daffy</td>
</tr>
<tr>
<td></td>
<td>222-22-2222</td>
<td>Pig</td>
<td>Porky</td>
</tr>
<tr>
<td></td>
<td>444-44-4444</td>
<td>Pig</td>
<td>Petunia</td>
</tr>
</tbody>
</table>
CREATE TABLE employees (  
    ssn CHAR(11),  
    last_name TEXT,  
    first_name TEXT  
)
Keys

• A key for an entity set is a set of attributes that uniquely identifies each entity
  – Two entities cannot have identical values for all the attributes in a key

• A key should be a *minimal* subset of attributes that satisfies the above
  – No subset of the attributes in a key is a unique identifier for an entity
• We can specify that a group of columns must be unique for each row by adding a UNIQUE constraint to the table
  – This will cause the DBMS to enforce the constraint that no two rows have the same values for both last_name and first_name
    • (Note this would mean that we couldn't have two employees with the same name, which is probably not what we really want—this is for illustration only.)

CREATE TABLE employees (  
  ssn CHAR(11),  
  last_name TEXT,  
  first_name TEXT,  
  UNIQUE (last_name, first_name)  
);
The DBMS will enforce the constraint and not allow us to insert or update a row so that it has the same last_name and first_name as another row.
  - But it doesn't keep up from entering rows with NULL values for the last_name and first_name.
    • Remember NULL ≠ NULL, so multiple rows could have NULL last_name or first_name.
NOT NULL

• We need to include a NOT NULL constraint on the columns if we want to exclude NULLs

```sql
CREATE TABLE employees (  
  ssn CHAR (11),  
  last_name TEXT NOT NULL,  
  first_name TEXT NOT NULL,  
  UNIQUE (last_name, first_name)  
);
```
Primary Keys

• One candidate key should be chosen as the primary key
  – PRIMARY KEY is basically the same as specifying UNIQUE and NOT NULL, except
    • There can be only one primary key per table
    • The DBMS optimizes the table for queries that use the primary key

```sql
CREATE TABLE employees (
    ssn CHAR(11),
    last_name TEXT,
    first_name TEXT,
    CONSTRAINT employees_key
      PRIMARY KEY (last_name, first_name)
);
```
Primary Keys

• If the primary key is a single column then a shorter version can be used

    CREATE TABLE employees (      
        ssn CHAR(11) PRIMARY KEY,      
        last_name TEXT,      
        first_name TEXT      
    );

• Most DBMSs allow duplicate rows unless constraints are specified, but this is rarely useful
  – Generally, every table should have a primary key
Foreign Keys

• Often, information in one table is linked to another table
  – The foreign key in the second table must match the columns of the primary key of the first table

<table>
<thead>
<tr>
<th>employees</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ssn</td>
<td>last_name</td>
</tr>
<tr>
<td>111-11-1111</td>
<td>Bunny</td>
</tr>
<tr>
<td>333-33-3333</td>
<td>Duck</td>
</tr>
<tr>
<td>222-22-2222</td>
<td>Pig</td>
</tr>
<tr>
<td>444-44-4444</td>
<td>Pig</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>addresses</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ssn</td>
<td>address</td>
</tr>
<tr>
<td>111-11-1111</td>
<td>1 Acme Lane</td>
</tr>
<tr>
<td>333-33-3333</td>
<td>5 Pond Ave</td>
</tr>
<tr>
<td>222-22-2222</td>
<td>3 Farm St</td>
</tr>
<tr>
<td>444-44-4444</td>
<td>3 Farm St</td>
</tr>
</tbody>
</table>
Foreign Keys

• A foreign key constraint can enforce the relationship

```sql
CREATE TABLE addresses (  
    ssn CHAR(11),  
    address TEXT,  
    FOREIGN KEY (ssn)  
    REFERENCES employees(ssn)
);
```

• The DBMS will not allow an insert or update on the addresses table if the value for ssn does not exist in the employees table
Foreign Keys

• If the foreign key is a single column then a shorthand version can be used

```sql
CREATE TABLE addresses(
    ssn CHAR(11) REFERENCES employees(ssn),
    address TEXT
);
```
Foreign Keys

• A foreign key constraint does not keep NULL values from being entered
  – Must either use NOT NULL constraints or PRIMARY KEY constraint

```sql
CREATE TABLE addresses(
  ssn CHAR(11) PRIMARY KEY
  REFERENCES employees(ssn),
  address TEXT
);
```

In this example, the ssn in the addresses table is both the primary key and a foreign key
Referential Integrity

- What happens if we attempt to delete a row from the employees table when there exists a row in the addresses table with that ssn?
  - Or change the ssn of an employees row?

- Four options:
  1. Don't allow the deletion of the row from employees
  2. Delete all rows from addresses that refer to the deleted row from employees
  3. Set the ssn of the addresses rows to some default value (which must exist in employees)
  4. Set the ssn of the addresses rows to NULL
Referential Integrity

- Default action is choice 1: do not allow the deletion
  - DBMS will give an error if a delete attempt is made
- Can change the action with ON DELETE

```
CREATE TABLE addresses (  
  ssn CHAR(11) PRIMARY KEY,  
  address TEXT,  
  FOREIGN KEY (ssn)  
  REFERENCES employees(ssn)  
  ON DELETE CASCADE  
);  
```

- ON DELETE CASCADE will cause all referencing rows in addresses to be deleted when the referenced row in employees is deleted
- Useful, but be careful!
Referential Integrity

- Can use ON DELETE SET DEFAULT to set the ssn to a default value when the referenced employee is deleted
  - Must also declare a default value for the ssn in addresses

```sql
CREATE TABLE addresses(
    ssn CHAR(11) PRIMARY KEY DEFAULT '000-00-0000',
    REFERENCES employees(ssn)
    ON DELETE SET DEFAULT,
    address TEXT
);
```

Note that because ssn has been declared a primary key, only one row can actually have the default value
Referential Integrity

• Can use ON DELETE SET NULL to set the ssn to NULL when the reference employee is deleted

```
CREATE TABLE addresses(
  ssn CHAR(11) PRIMARY KEY
  REFERENCES employees(ssn)
  ON DELETE SET NULL,
  address TEXT
);
```

Note that because ssn has been declared a primary key, the ssn cannot actually be set to NULL, so the delete from employees will fail with an error
Referential Integrity

• The same issues when deleting occur when updating
  – Can use ON UPDATE to specify alternate actions
    • The default action (for both update and delete) is NO ACTION, which is to not allow the update or delete

```sql
CREATE TABLE addresses(
    ssn CHAR(11) PRIMARY KEY
    REFERENCES employees(ssn)
    ON DELETE CASCADE
    ON UPDATE NO ACTION,
    address TEXT
);
```
A many-to-many relationship set is implemented using a mapping table with foreign keys to the primary keys of the entity tables.
A many-to-many relationship set is implemented using a mapping table with foreign keys to the primary keys of the entity tables.
Many-to-Many

CREATE TABLE employees (  
    ssn CHAR(11) PRIMARY KEY,  
    name TEXT NOT NULL  
);  

CREATE TABLE departments (  
    did INTEGER PRIMARY KEY,  
    name TEXT NOT NULL  
);  

CREATE TABLE works_in (  
    ssn CHAR(11) REFERENCES employees(ssn),  
    did INTEGER REFERENCES departments(did),  
    since INTEGER NOT NULL,  
    PRIMARY KEY (ssn, did)  
);  

• Note this can be expanded to relationships between more than two tables
A one-to-many relationship set can also be implemented using a mapping table with foreign keys to the primary keys of the entity tables. With the additional constraint that the foreign key to the "many" entity set be unique in the mapping table.
One-to-Many

- In this example, assume the relationship from departments to employees is one-to-many
  - So departments can have multiple employees but employees can only work in one department
- So the unique constraint needs to be on the ssn column of works_in (not the did column)
One-to-Many

CREATE TABLE employees (  
  ssn  CHAR(11) PRIMARY KEY,  
  name TEXT NOT NULL  
);  

CREATE TABLE departments (  
  did INTEGER PRIMARY KEY,  
  name TEXT NOT NULL  
);  

CREATE TABLE works_in (  
  ssn  CHAR(11) REFERENCES employees(ssn) UNIQUE,  
  did INTEGER REFERENCES departments(did),  
  since INTEGER NOT NULL,  
  PRIMARY KEY (ssn, did)  
);  

• In this example, assume the relationship from departments to employees is one-to-many  
  – So departments can have multiple employees but employees can only work in one department  
• So the unique constraint needs to be on the ssn column of works_in (not the did column)
A one-to-one mapping table would have unique constraints on both foreign keys.
### One-to-One

<table>
<thead>
<tr>
<th>employees</th>
<th>manages</th>
<th>departments</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssn</td>
<td>name</td>
<td>did</td>
</tr>
<tr>
<td>111-11-1111</td>
<td>Bunny, Bugs</td>
<td>111-11-1111</td>
</tr>
<tr>
<td>333-33-3333</td>
<td>Duck, Daffy</td>
<td>333-33-3333</td>
</tr>
<tr>
<td>222-22-2222</td>
<td>Pig, Porky</td>
<td>444-44-4444</td>
</tr>
<tr>
<td>444-44-4444</td>
<td>Pig, Petunia</td>
<td></td>
</tr>
</tbody>
</table>

- A one-to-one mapping table would have unique constraints on both foreign keys
One-to-One

CREATE TABLE employees (  
    ssn CHAR(11) PRIMARY KEY,  
    name TEXT NOT NULL  
);  

CREATE TABLE departments (  
    did INTEGER PRIMARY KEY,  
    name TEXT NOT NULL  
);  

CREATE TABLE manages (  
    ssn CHAR(11) REFERENCES employees(ssn) UNIQUE,  
    did INTEGER REFERENCES departments(did) UNIQUE,  
    since INTEGER NOT NULL,  
    PRIMARY KEY (ssn, did)  
);  

• Note that having UNIQUE constraints on the individual columns is different than having a UNIQUE constraint on the combination
Roles

- We can also have a reference from a table to itself
Self-Reference

We can again use a mapping table
  – Potentially with UNIQUE constraints if one-to-many or one-to-one is desired

<table>
<thead>
<tr>
<th>employees</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ssn</td>
<td>name</td>
</tr>
<tr>
<td>111-11-1111</td>
<td>Bunny, Bugs</td>
</tr>
<tr>
<td>333-33-3333</td>
<td>Duck, Daffy</td>
</tr>
<tr>
<td>222-22-2222</td>
<td>Pig, Porky</td>
</tr>
<tr>
<td>444-44-4444</td>
<td>Pig, Petunia</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>reports_to</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>supervisor</td>
<td>underling</td>
</tr>
<tr>
<td>111-11-1111</td>
<td>333-33-3333</td>
</tr>
<tr>
<td>333-33-3333</td>
<td>222-22-2222</td>
</tr>
<tr>
<td>444-44-4444</td>
<td>111-11-1111</td>
</tr>
</tbody>
</table>
Self-Reference

CREATE TABLE employees (  
    ssn CHAR(11) PRIMARY KEY,  
    name TEXT NOT NULL  
);  

CREATE TABLE reports_to (  
    supervisor CHAR(11) REFERENCES employees(ssn),  
    underling CHAR(11) REFERENCES employees(ssn) UNIQUE,  
    PRIMARY KEY (supervisor, underling)  
);  

• We can again use a mapping table  
  – Potentially with UNIQUE constraints if one-to-many or one-to-one is desired
### One-to-Many

<table>
<thead>
<tr>
<th>employees</th>
<th>departments</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssn</td>
<td>did</td>
</tr>
<tr>
<td>111-11-1111</td>
<td>5</td>
</tr>
<tr>
<td>333-33-3333</td>
<td>4</td>
</tr>
<tr>
<td>222-22-2222</td>
<td>4</td>
</tr>
<tr>
<td>444-44-4444</td>
<td>7</td>
</tr>
</tbody>
</table>

- For one-to-many, we can use a foreign key in the "many" table instead of a separate mapping table
  - Often preferred as it eliminates a table and a JOIN in queries
### One-to-One

<table>
<thead>
<tr>
<th>employees</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ssn</td>
<td>name</td>
<td>did</td>
<td>since</td>
<td></td>
</tr>
<tr>
<td>111-11-1111</td>
<td>Bunny, Bugs</td>
<td>5</td>
<td>2001</td>
<td></td>
</tr>
<tr>
<td>333-33-3333</td>
<td>Duck, Daffy</td>
<td>4</td>
<td>2001</td>
<td></td>
</tr>
<tr>
<td>444-44-4444</td>
<td>Pig, Petunia</td>
<td>7</td>
<td>2002</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>departments</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>did</td>
<td>name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>HR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>IT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Sales</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- One-to-one is a special case of one-to-many with an extra UNIQUE constraint on the foreign key
  - In this example, that means the did column in employees should be unique
- Note that many-to-many requires a separate mapping table
Self-Reference

If the self-reference is one-to-many or one-to-one then a foreign key in the table to itself can be used.

<table>
<thead>
<tr>
<th>ssn</th>
<th>name</th>
<th>supervisor</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-11-1111</td>
<td>Bunny, Bugs</td>
<td>444-44-4444</td>
</tr>
<tr>
<td>333-33-3333</td>
<td>Duck, Daffy</td>
<td>111-11-1111</td>
</tr>
<tr>
<td>222-22-2222</td>
<td>Pig, Porky</td>
<td>333-33-3333</td>
</tr>
<tr>
<td>444-44-4444</td>
<td>Pig, Petunia</td>
<td>NULL</td>
</tr>
</tbody>
</table>
Total Participation

• NOT NULL constraints can sometimes be used to require total participation of an entity set in a relationship set
  – For example, if every department must have a manager then a NOT NULL constraint could be added to the department's manager column

```sql
CREATE TABLE departments (  
did INTEGER PRIMARY KEY,  
nname TEXT NOT NULL,  
manager CHAR(11) REFERENCES employees(ssn) NOT NULL
);
```

• In general, however, participation constraints often require more expensive table constraints or assertions to enforce
  – We'll talk about these later in the semester, but they're often a bad idea as they can severely impact performance
Weak Entity Sets

- Weak entity sets can be viewed as a special case of a one-to-many relationship.

Diagram:
- Employees
  - ssn
  - name
- Insurance
  - dname
  - age
- Dependents
Weak Entity Sets

CREATE TABLE employees (  
    ssn CHAR(11) PRIMARY KEY,  
    name TEXT NOT NULL  
);  

CREATE TABLE dependents (  
    ssn CHAR(11) REFERENCES employees(ssn) NOT NULL,  
    dname TEXT NOT NULL,  
    age INTEGER NOT NULL,  
    PRIMARY KEY (ssn, dname)  
);  

• The foreign key reference is in the "many" side (dependents) to the "one" side (employees)  
• The foreign key and key attributes have NOT NULL constraints  
• The combination of the foreign key and the key attribute are declared as a PRIMARY KEY
Class Hierarchies

• Two options:
  1. A parent table with child tables for the subclasses
     • Each child table has a foreign key reference to the parent table, typically with a one-to-one relationship
  2. Child tables with no parent table

• Option 1 is generally preferred
  – Option 2 disallows having a record which doesn't fit into any of the child subclasses
  – Option 2 also make operations like selecting all employees more difficult
Aggregation

• First, create tables for the aggregate part

CREATE TABLE projects (
    pid INTEGER PRIMARY KEY,
    budget INTEGER NOT NULL
);

CREATE TABLE departments (
    did INTEGER PRIMARY KEY,
    name TEXT NOT NULL
);

CREATE TABLE sponsors (
    pid INTEGER REFERENCES projects(pid),
    did INTEGER REFERENCES departments(did),
    since INTEGER NOT NULL,
    PRIMARY KEY (pid, did)
);
Aggregation

• Then, create the table for the relationship to the aggregate, using the primary key of the aggregate's relation as a foreign key

CREATE TABLE employees (  
  ssn CHAR(11) PRIMARY KEY,  
  name TEXT NOT NULL  
);

CREATE TABLE monitors (  
  ssn CHAR(11) REFERENCES employees(ssn) NOT NULL,  
  pid INTEGER NOT NULL,  
  did INTEGER NOT NULL,  
  until INTEGER NOT NULL,  
  PRIMARY KEY (ssn, pid, did),  
  FOREIGN KEY (pid, did) REFERENCES sponsors(pid, did)  
);
ALTER TABLE

- Constraints can be added or removed after table creation by using ALTER TABLE

    CREATE TABLE employees (  
        ssn CHAR(11) PRIMARY KEY,  
        name TEXT  
    );

    ALTER TABLE employees ALTER COLUMN name SET NOT NULL;

- However, adding a constraint after data has been inserted will fail if the existing data violates the constraint
CREATE TABLE customers (  
cid INTEGER PRIMARY KEY,  
name TEXT NOT NULL  
);
Example

CREATE TABLE orders ( 
    nbr INTEGER PRIMARY KEY 
);

CREATE TABLE orders ( 
    nbr INTEGER PRIMARY KEY,
    cid INTEGER REFERENCES customers(cid) NOT NULL
 );
Example

CREATE TABLE products ( isbn TEXT PRIMARY KEY );
CREATE TABLE includes (  
nbr INTEGER REFERENCES orders(nbr),  
isbn TEXT REFERENCES products(isbn),  
quantity INTEGER NOT NULL,  
PRIMARY KEY (nbr, isbn)  
)