This lecture continues the discussion of encoding relationship sets into tables, but this time considering how things work in the presence of participation and key constraints.

**Relationship Sets**

**Encoding relationship sets**

When discussing the encoding of relationship sets, it is useful to distinguish relationships that are:
- One-to-One
- One-to-Many
- Many-to-Many

We will use examples based on the book "Fundamentals of Database Management Systems" by Mark. L. Gillenson.

These examples are based on a hypothetical hardware supplier *General Hardware* that sells hardware products in quantity to retail outlets.

This ER diagram indicates that every customer of *General Hardware* has a unique salesperson responsible for that account.
Sample entity sets

<table>
<thead>
<tr>
<th>staffId</th>
<th>name</th>
<th>hireYear</th>
</tr>
</thead>
<tbody>
<tr>
<td>137</td>
<td>Baker</td>
<td>1995</td>
</tr>
<tr>
<td>186</td>
<td>Adams</td>
<td>2001</td>
</tr>
<tr>
<td>204</td>
<td>Dickens</td>
<td>1998</td>
</tr>
<tr>
<td>361</td>
<td>Carlyle</td>
<td>2004</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>customerId</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1121</td>
<td>Bunnings</td>
<td>Subiaco</td>
</tr>
<tr>
<td>1122</td>
<td>Bunnings</td>
<td>Claremont</td>
</tr>
<tr>
<td>1211</td>
<td>Mitre 10</td>
<td>Myaree</td>
</tr>
<tr>
<td>1244</td>
<td>Mitre 10</td>
<td>Joondalup</td>
</tr>
<tr>
<td>1345</td>
<td>Joe’s Hardware</td>
<td>Nedlands</td>
</tr>
<tr>
<td>1399</td>
<td>NailsRUs</td>
<td>Jolimont</td>
</tr>
</tbody>
</table>

The relationship set

Now suppose that Baker is the account manager for both Bunnings outlets, Adams for both Mitre 10 outlets, Dickens for Joe’s Hardware and Carlyle for NailsRUs.

Then we could have a separate table SellsTo, similar to the works table from last lecture.

<table>
<thead>
<tr>
<th>staffId</th>
<th>customerId</th>
</tr>
</thead>
<tbody>
<tr>
<td>137</td>
<td>1121</td>
</tr>
<tr>
<td>137</td>
<td>1122</td>
</tr>
<tr>
<td>186</td>
<td>1211</td>
</tr>
<tr>
<td>186</td>
<td>1244</td>
</tr>
<tr>
<td>204</td>
<td>1345</td>
</tr>
<tr>
<td>361</td>
<td>1399</td>
</tr>
</tbody>
</table>

What is the key?

In the works relation, we needed the combination of the staff number and department number to form a key because the relationship was many-to-many.

However here the relationship is one-to-many and so each customerId can only appear once in the table, and so we can use customerId as the key.

But what this suggests to us is that this table is unnecessary because we could incorporate this information into the Customer relation.

This permits us to eliminate one table with the resulting simplification of queries.

The one-to-many approach

CREATE TABLE customer  
    (customerId INT, 
     PRIMARY KEY (customerId), 
     name VARCHAR(20), 
     address VARCHAR(40), 
     accountMgr INT  
    );

We incorporate an additional field into customer which contains the staffId for the salesperson responsible for that company’s account.

This field is called a foreign key because it is the key value for a different table.
**Relationship Sets**

### The new entity set

<table>
<thead>
<tr>
<th>customerId</th>
<th>name</th>
<th>address</th>
<th>accountMgr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1121</td>
<td>Bunnings</td>
<td>Subiaco</td>
<td>137</td>
</tr>
<tr>
<td>1122</td>
<td>Bunnings</td>
<td>Claremont</td>
<td>137</td>
</tr>
<tr>
<td>1211</td>
<td>Mitre 10</td>
<td>Myaree</td>
<td>186</td>
</tr>
<tr>
<td>1244</td>
<td>Mitre 10</td>
<td>Joondalup</td>
<td>186</td>
</tr>
<tr>
<td>1345</td>
<td>Joe’s Hardware</td>
<td>Nedlands</td>
<td>204</td>
</tr>
<tr>
<td>1399</td>
<td>NailsRUs</td>
<td>Jolimont</td>
<td>361</td>
</tr>
</tbody>
</table>

*GF Royle 2006-8, N Spadaccini 2008*

### Participation Constraints

The advantage of this second approach to the one-to-many relationship is that it allows us to enforce the *participation constraints* that every customer must have an account manager — we simply have to add **NOT NULL** to that field.

```sql
CREATE TABLE customer ( 
  customerId INT,
  PRIMARY KEY (customerId),
  name VARCHAR(20),
  address VARCHAR(40),
  accountMgr INT NOT NULL
);
```

*GF Royle 2006-8, N Spadaccini 2008*

### One-to-One

Suppose that every salesperson has an office and every office houses at most one salesperson.

### Choices

We have a number of different choices

- A single table combining the salesperson and office information
- Incorporating the office information as a foreign key in the salesperson entity set
- Incorporating the salesperson information as a foreign key in the office entity set

What are the pros and cons of these options?
Is there total participation?

The crucial factor here is whether there is total participation of the entities in the relationship set — the diagram does not show total participation.

If the relationship is one-to-one with total participation, so that every salesperson has an office and every office has a salesperson, then we can simply incorporate all of the “office” information into the SalesPerson relation.

In this situation, there would be two candidate keys, namely the staffId and the roomId.

(A bad design)

In general, this is a bad design because the relationship is not intrinsically a one-to-one relationship with total participation.

What happens if a salesperson resigns? Then their office becomes vacant, but there is no way of recording this with the current structure.

If a relationship really is 1-1 with total participation, then it could (would?) be modelled as the attributes of an entity set in the ER diagram, rather than as a relationship set.

(The table)

CREATE TABLE salesPerson (  
staffId INT,  
PRIMARY KEY (staffId),  
name VARCHAR(20),  
hireYear DATE,  
roomId INT,  
phone INT,  
location VARCHAR(20)  
);

(Other choices)

The other two choices are to incorporate the office number as a foreign key in the SalesPerson entity set, or the incorporate the staff number as a foreign key in the Office entity set.

The correct decision here follows from the business logic — we should include the office number as a NOT NULL foreign key in the SalesPerson entity set because each salesperson must be assigned an office.

Empty offices are then simply entries in the Office table that do not appear in the SalesPerson table, and cause no problems.
Foreign Keys

Consider again the works table which relates lecturers and departments.

CREATE TABLE works (  
  staff_id INT,  
  dept_no INT,  
  percentage INT,  
  PRIMARY KEY (staff_id, dept_no)  
);  

The fields staff_id and dept_no are foreign keys that refer to the fields of the same name in the tables lecturer and department.

Although we know that these fields are meant to refer to the other tables, the DBMS does not.

This means that there is nothing stopping us from entering data into the works table that has a staff-id that does not actually correspond to any actual staff member, or a dept-no that does not correspond to a department.

INSERT INTO works VALUES(-1,12,50);

This is called a violation of referential integrity — in other words, the references from one table to another are corrupt.

Foreign key support

SQL permits the user to declare that a certain field is a foreign key that references another table.

CREATE TABLE works (  
  staff_id INT,  
  dept_no INT,  
  percentage INT,  
  PRIMARY KEY (staff_id, dept_no),  
  FOREIGN KEY (staff_id) REFERENCES lecturer (staff_id),  
  FOREIGN KEY (dept_no) REFERENCES department (dept_no)  
);  

In MySQL, only the InnoDB storage engine provides foreign key support and this must be declared when the table is created.

CREATE TABLE works (  
  staff_id INT,  
  dept_no INT,  
  percentage INT,  
  PRIMARY KEY (staff_id, dept_no),  
  FOREIGN KEY (staff_id) REFERENCES lecturer (staff_id),  
  FOREIGN KEY (dept_no) REFERENCES department (dept_no)  
) ENGINE = InnoDB;
Referential Integrity

Once the DBMS knows that the fields really are foreign keys it can maintain referential integrity by forbidding operations that would violate referential integrity.

```sql
mysql> INSERT INTO works
    -> VALUES(6,1,100);
ERROR 1452 (23000): Cannot add or update a child row:
    a foreign key constraint fails ('university/works',
    CONSTRAINT 'works_ibfk_1'
    FOREIGN KEY ('staff_id')
    REFERENCES 'lecturer' ('staff_id'))
```

This operation is forbidden because the value “6” is not a legitimate staff number according to the lecturer table.

Deletion

What should happen to a row containing a foreign key if the thing that it refers to is deleted?

Suppose for example that Adam Zoot leaves the university and we want to remove his records from the database.

```sql
mysql> DELETE FROM lecturer WHERE name = "Adam Zoot";
ERROR 1451 (23000): Cannot delete or update a parent row:
    a foreign key constraint fails ('university/works',
    CONSTRAINT 'works_ibfk_1'
    FOREIGN KEY ('staff_id')
    REFERENCES 'lecturer' ('staff_id'))
```

By default this action is forbidden because it would leave the works table with a dangling reference — a foreign key that referred to a non-existent lecturer.

Cascade

In this situation, it is clear that we want to delete the associated records in the works table because they are no longer relevant.

```sql
CREATE TABLE works (  
    staff_id INT,
    dept_no INT,
    percentage INT,
    PRIMARY KEY (staff_id, dept_no),
    FOREIGN KEY (staff_id) REFERENCES lecturer (staff_id)
    ON DELETE CASCADE,
    FOREIGN KEY (dept_no) REFERENCES department (dept_no)
) ENGINE = InnoDB;
```

The ON DELETE CASCADE says that if the parent row is deleted then the consequences of this deletion should cascade through to this table.

Another option

Another option for a foreign key relationship is ON DELETE SET NULL which will cause the corresponding columns to be set to NULL rather than deleted entirely.

This option would be appropriate for example in the Customer relation from the General Hardware database where the foreign key accountMgr is set to NULL if the salesperson leaves.