This lecture covers the process of implementing an ER diagram as an actual relational database.

This involves converting the various entity sets and relationship sets into tables (i.e. relations) in such a way that the DBMS can check and enforce the participation and key constraints.

This lecture could equally well be called “From ER diagrams to relational tables”.

Entity Sets

Implementing entity sets is usually straightforward.

Each attribute of the entity set becomes a column of a relational table, and the only real choice needed is to select a type for that column.

```
CREATE TABLE lecturer (  
    staff_id INT,  
    name VARCHAR(20),  
    rank VARCHAR(20)  
);
```
## Entity Sets

### The staff ID

We have chosen the numeric type INT for the `staff-id` — this choice however will depend on the business logic for that particular organization.

Prior to 2005, UWA student “numbers” were allowed to contain the non-numeric character X so that something like

```
9712303X
```

would be a valid student number.

Even today, most staff numbers at UWA start with zeros, such as

```
00097382
```

In both of these situations, we would need to use one of the string types.

---

### Specifying the key

When creating a table, it is possible (and highly recommended) to specify the key for the table.

```sql
CREATE TABLE lecturer (
    staff_id INT,
    PRIMARY KEY (staff_id),
    name VARCHAR(20),
    rank VARCHAR(20)
);
```

The main consequence of this specification is that the DBMS will ensure that this relation never contains duplicate rows.

---

### The special value NULL

Sometimes the rows in a relational database table can only be partly specified.

For example, in the President database table we cannot enter the date-of-death for the most recent US presidents. In this situation there is a special value NULL which essentially means that “this value is not yet specified”.

However there are many situations where a field should not be permitted to have a NULL value, and in particular a key value can never be NULL.

Declaring a field to be a primary key means that the database enforces the restriction that it is both unique and not NULL.

---

### Values that cannot be NULL

In many other situations, the business logic dictates that a certain attribute cannot be NULL and so there is a way of specifying this restriction separately from being a key.

```sql
CREATE TABLE lecturer (
    staff_id INT,
    PRIMARY KEY (staff_id),
    name VARCHAR(20) NOT NULL,
    rank VARCHAR(20)
);
```
Any old numbers

Frequently the number used for a primary key has no meaning other than to uniquely identify that row — provided it is different from the other numbers we don’t actually care what it is.

We can get the DBMS to assign the numbers for us automatically by modifying the table appropriately.

```
CREATE TABLE lecturer (
    staff_id INT AUTO_INCREMENT,
    PRIMARY KEY (staff_id),
    name VARCHAR(20) NOT NULL,
    rank VARCHAR(20)
);
```

Using auto-increment

If a column is auto-increment then specifying its value is optional when inserting data into the table.

- If the number is specified (and valid) then it is used
  ```
  INSERT INTO lecturer
  VALUES(1454,"John Smith","Associate Professor");
  ```
- If the number is not specified then a unique number is generated.
  ```
  INSERT INTO lecturer
  VALUES(NULL,"Gill Lee","Professor");
  ```

(Yes, it does seem odd to specify NULL for a column that is explicitly not permitted to be NULL.)

The rank

So far we have specified a string type, namely VARCHAR(20) for the rank of the lecturer.

Although this would work, it is unnecessarily error-prone because we know in advance that there are only five different ranks — Associate Lecturer, Lecturer, Senior Lecturer, Associate Professor and Professor. Ideally we would like a type that could only take those values, so that a typing mistake like

```
INSERT INTO lecturer
VALUES(1454,"John Smith","Asscoiate Professor");
```

could be automatically detected.

Enumerated Types

SQL has an enumerated type facility that allows the values that a field can take to be restricted to values from a specified list.

```
CREATE TABLE lecturer (
    staff_id INT AUTO_INCREMENT,
    PRIMARY KEY (staff_id),
    name VARCHAR(20) NOT NULL,
    rank ENUM('AL', 'L', 'SL', 'AP', 'P')
);
```

This specifies that the value of rank can only be one of the five possible strings AL, L, SL, AP or P.
Encoding Relationship Sets

The simplest “plain vanilla” situation is a binary relationship with no constraints.

This is a variant of the ER diagram previously discussed which relates lecturers to the departments in which they work.

A direct mapping

This relationship set maps directly to its own table.

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

Here the fields staff_id and dept_no refer to the key values for the tables Lecturer and Department respectively, while the percentage is the relationship attribute.

Sample entity set data

mysql> select * from lecturer;
<table>
<thead>
<tr>
<th>staff_id</th>
<th>name</th>
<th>rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Gill Lee</td>
<td>P</td>
</tr>
<tr>
<td>2</td>
<td>John Smith</td>
<td>AP</td>
</tr>
<tr>
<td>3</td>
<td>Adam Zoot</td>
<td>AL</td>
</tr>
<tr>
<td>4</td>
<td>Bill Jackson</td>
<td>L</td>
</tr>
<tr>
<td>5</td>
<td>Jessica Ormerod</td>
<td>SL</td>
</tr>
</tbody>
</table>
+----------+-----------------+------+
5 rows in set (0.00 sec)

mysql> select * from department;
<table>
<thead>
<tr>
<th>dept_no</th>
<th>name</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Computer Science</td>
<td>2716</td>
</tr>
<tr>
<td>2</td>
<td>Mathematics</td>
<td>1574</td>
</tr>
<tr>
<td>3</td>
<td>Physics</td>
<td>2746</td>
</tr>
<tr>
<td>4</td>
<td>Chemistry</td>
<td>9716</td>
</tr>
<tr>
<td>5</td>
<td>Geology</td>
<td>1458</td>
</tr>
</tbody>
</table>
+---------+------------+--------+
5 rows in set (0.00 sec)

mysql> select * from works;
<table>
<thead>
<tr>
<th>staff_id</th>
<th>dept_no</th>
<th>percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>80</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>50</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>50</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>70</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>100</td>
</tr>
</tbody>
</table>
+----------+---------+------------+
8 rows in set (0.00 sec)

Each record (row, tuple) gives information about how a particular lecturer relates to a particular department.
Who works in physics?

How do we use this information to find out who works in Physics?

We need to

- Look in the `department` table to find out the department number of Physics.
- Look in the `works` table to find out staff numbers of the people working in departments with that particular number.
- Look in the `lecturer` table to find the names of the lecturers with those staff numbers.

A simpler version

First we'll just find out the staff numbers of those who work in Physics.

```sql
mysql> SELECT staff_id FROM works, department WHERE
   
   -> works.dept_no = department.dept_no
   
   -> AND
   
   -> department.name = "Physics";
```

```plaintext
+----------+
| staff_id |
+----------+
| 2        |
| 3        |
+----------+
2 rows in set (0.00 sec)
```

What does this mean?

The `FROM` clause implies that we are (notionally) considering *every possible combination* of rows from the two tables `works` and `department`.

Most of these rows are useless . . .

Of course most of these combinations of rows are useless because they contain things that are *not related* to each other.

The conditions, specified in the `WHERE` clause first specify the row-combinations that make sense in that are both talking about the same department.

```
WHERE works.dept_no = department.dept_no
```
... until we select just those we want ...

Then we add the additional condition that we want to know only about Physics.

```
AND
department.name = "Physics";
```

```
+----------+---------+------------+---------+---------+-------+
<table>
<thead>
<tr>
<th>staff_id</th>
<th>dept_no</th>
<th>percentage</th>
<th>dept_no</th>
<th>name</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>100</td>
<td>3</td>
<td>Physics</td>
<td>2746</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>50</td>
<td>3</td>
<td>Physics</td>
<td>2746</td>
</tr>
</tbody>
</table>
+----------+---------+------------+---------+---------+-------+
2 rows in set (0.03 sec)
```

... and finally get the answer!

This collection of rows has many fields (including two called dept_no) but we are only interested in the staff_id.

```
+----------+
| staff_id |
+----------+
| 2        |
| 3        |
+----------+
2 rows in set (0.01 sec)
```

But who are they?

Of course a user is immediately likely to ask who the lecturers are, not just what their staff numbers are, which means that we have to involve the lecturer table as well.

```
mysql> SELECT lecturer.name FROM lecturer, works, department
   -> WHERE
   -> lecturer.staff_id = works.staff_id
   -> AND
   -> works.dept_no = department.dept_no
   -> AND
   -> department.name = "Physics";
```

```
+------------+
<table>
<thead>
<tr>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Smith</td>
</tr>
<tr>
<td>Adam Zoot</td>
</tr>
</tbody>
</table>
+------------+
```

How are queries built?

A fundamental reason for the success of SQL and RDBMS is that queries like these can be expressed formally using either relational algebra or relational calculus and analysed to produce a reasonable evaluation strategy.

This allows the user to create complex queries involving multiple tables and have the DBMS work out the best way to actually perform that query.