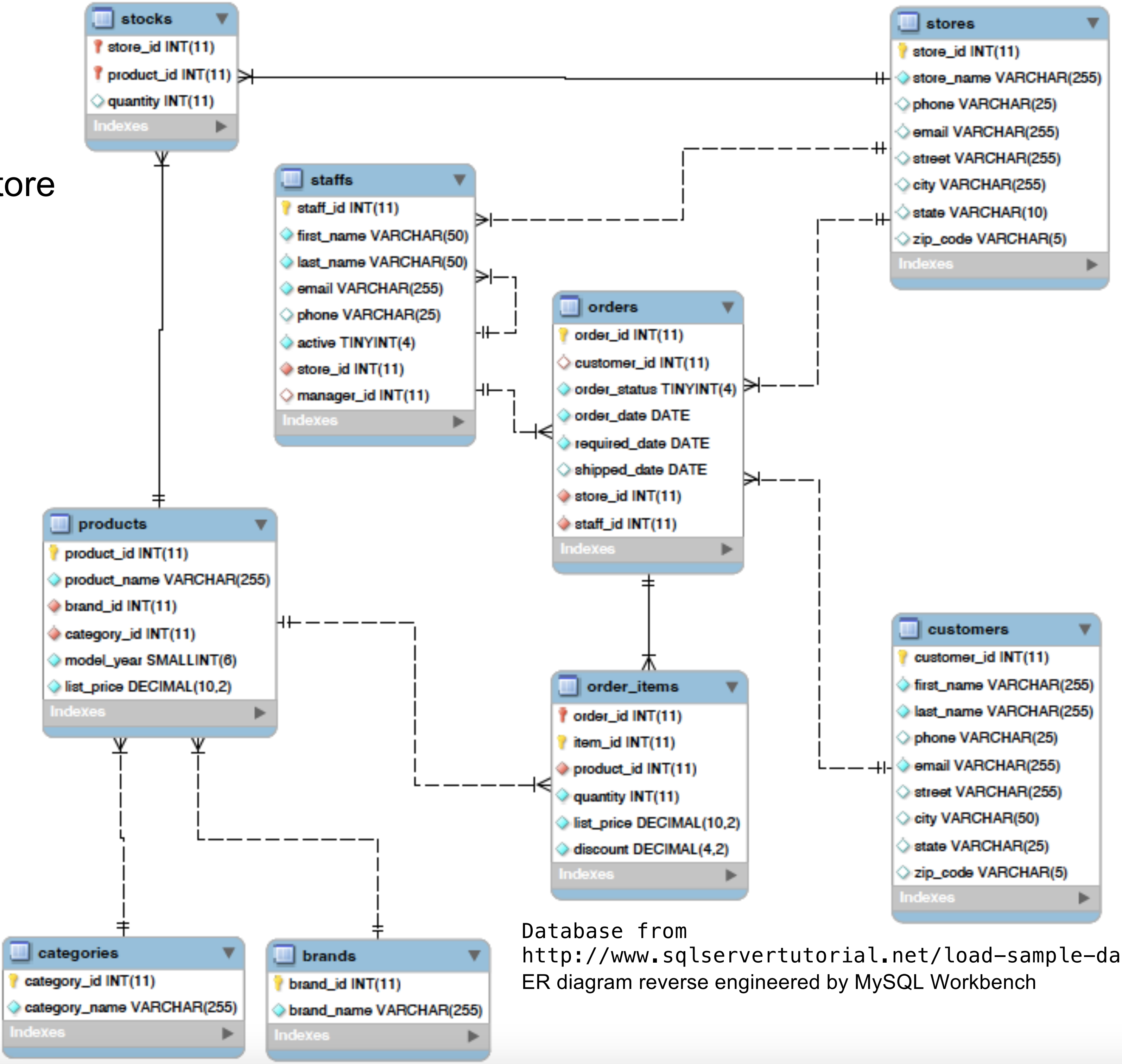
Relations

- Degree – number of entity types that participate
  - Unary – one
  - Binary – two
  - Ternary – three

- Cardinality
  - The number of entities that participate in the relation

Bicycle store database



stocks
- store_id INT(11)
- product_id INT(11)
- quantity INT(11)

Indexes

stores
- store_id INT(11)
- store_name VARCHAR(255)
- phone VARCHAR(25)
- email VARCHAR(255)
- street VARCHAR(255)
- city VARCHAR(255)
- state VARCHAR(10)
- zip_code VARCHAR(5)

Indexes

staffs
- staff_id INT(11)
- first_name VARCHAR(50)
- last_name VARCHAR(50)
- email VARCHAR(255)
- phone VARCHAR(25)
- active TINYINT(4)
- store_id INT(11)
- manager_id INT(11)

Indexes

orders
- order_id INT(11)
- customer_id INT(11)
- order_status TINYINT(4)
- order_date DATE
- required_date DATE
- shipped_date DATE
- store_id INT(11)
- staff_id INT(11)

Indexes

products
- product_id INT(11)
- product_name VARCHAR(255)
- brand_id INT(11)
- category_id INT(11)
- model_year SMALLINT(6)
- list_price DECIMAL(10,2)

Indexes

order_items
- order_id INT(11)
- item_id INT(11)
- product_id INT(11)
- quantity INT(11)
- list_price DECIMAL(10,2)
- discount DECIMAL(4,2)

Indexes

customers
- customer_id INT(11)
- first_name VARCHAR(255)
- last_name VARCHAR(255)
- phone VARCHAR(25)
- email VARCHAR(255)
- street VARCHAR(255)
- city VARCHAR(50)
- state VARCHAR(25)
- zip_code VARCHAR(5)

Indexes

categories
- category_id INT(11)
- category_name VARCHAR(255)

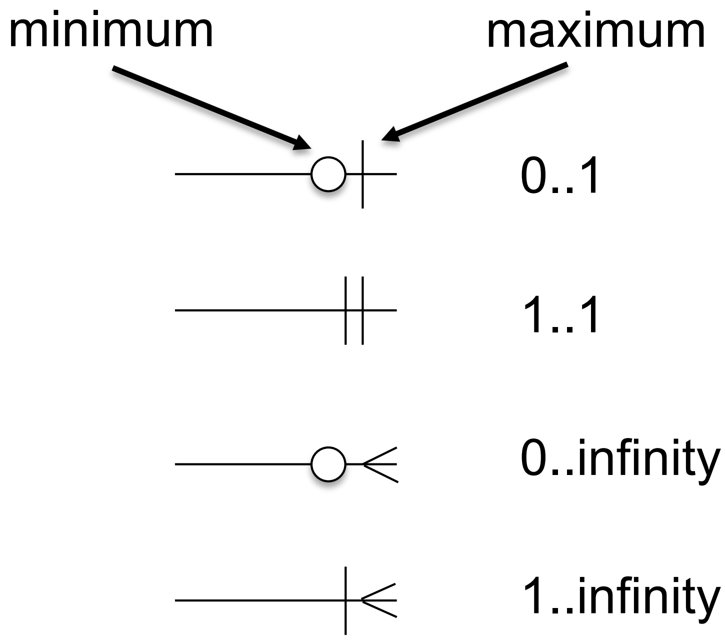
Indexes

brands
- brand_id INT(11)
- brand_name VARCHAR(255)

Indexes

Database from
http://www.sqlservertutorial.net/load-sample-database/
ER diagram reverse engineered by MySQL Workbench

ERM symbols – relations



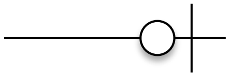
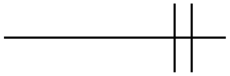
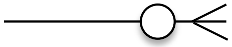
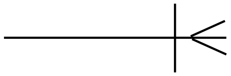
minimum

maximum

0..1	At most one
1..1	Exactly one
0..infinity	0 or more
1..infinity	1 or more

Typically connect the relation edges between the matching primary and foreign keys

ERM symbols – relations

	Optional one
	Mandatory one
	Optional many
	Mandatory many

Typically connect the relation edges between the matching primary and foreign keys

Data Modeling

- Defines
  - which entities you have,
  - how they are grouped,
  - the relation between entities, and
  - the cardinality of the relations.

- Come from your analysis of the business
  - Derived from explicit and implicit business rules

Creating tables

-
  ```
  Create table  [if not exists]  <tablename> (
      <columnName1>  <datatype>,
      <columnName2>  <datatype>
  );
  ```

- Modifiers for after the data type:
  - Not null – prevent NULL values from being stored
  - Default X – set default value to X on inserts
  - Auto_increment – designates an increment field for surrogate keys (but doesn’t automatically make the column an key)

Relational keys

- Primary key
  - An attribute (or combination of attributes) that uniquely identifies each row in a relation
    - The choice of primary key may not be unique

- Composite key
  - A primary key that consists of more than one attribute.

DALHOUSIE UNIVERSITY
Inspiring Minds

Relational keys

- Foreign key
  - An attribute in a relation that serves as the primary key of another relation in the same database.

- Surrogate key
  - A serial number or other system assigned primary key for a relation
  - Often created to replace
    - a complex or highly-composite primary key
    - an expensive primary key (often big strings)
    - a primary key that could be re-used over time

SQL Context

- Endings to the “create table” command:
  - Primary key (<id> [, <id>, <id>, …]) – defines the primary key of the table, basic or composite
  - Foreign key (<id>) references <table> (<key>) – defines field “id” as a foreign key in the current table that maps to primary key <key> in table <table>
  - Check <field condition> -- ensures that data meets a criterion
    - Eg field condition could be: Age >= 20
      to ensure that all ages are 20 or more in the table.
  - Can give these endings a name:
    - Constraint <name> <ending from above>

DALHOUSIE
UNIVERSITY
Inspiring Minds

SQL example

```
Create table sample3 (
    id int not null,
    name char(10),
    primary key (id)
);

create table sample4 (
    info int not null,
    value char(10),
    id int,
    primary key (info),
    foreign key (id) references sample3 (id)
);
```

SQL context

- Eg.

```
Create table employees (
    employeeNumber int not null,
    lastName char(50) not null,
    firstName char(20) not null,
    age int,
    primary key (employeeNumber),
    check age >= 15
);
```

Deleting tables

- Drop table <tablename>;

Converting to a database

- Relation – a named two-dimensional table of data
- Properties of relations
  - Each relation has a unique name.
  - An entry at the intersection of each row and column is atomic. There are no multivalued attributes.
  - Each row is unique.
  - Each attribute / column within a table has a unique name
  - The sequence of columns is insignificant.
  - The sequence of rows is insignificant.

Conversion steps

1. Map regular entities
2. Map weak entities
3. Map binary relations
4. Map associative entities
5. Map unary relations
6. Map ternary or more complex relations

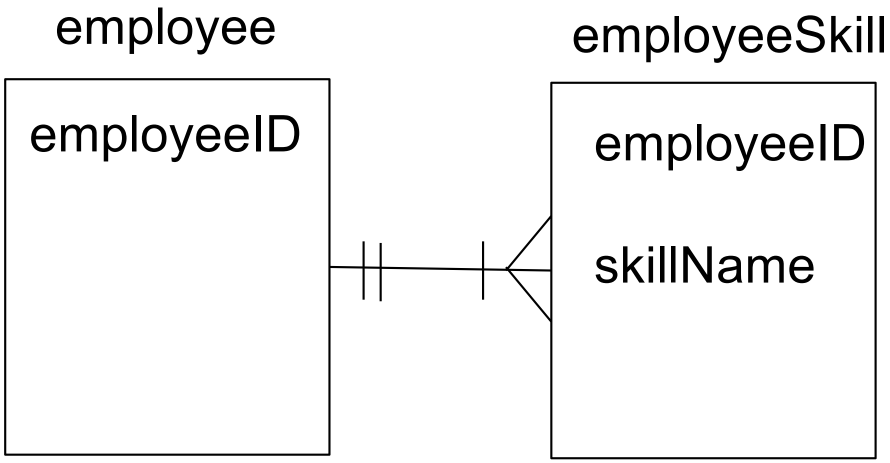
Map regular entities

- Basic attributes become table columns
- Composite attributes only have their subcomponents stored
  - Eg. “address” is composite, so we would store street address, city, province, country, postal code individually but not ”address” itself.
- Multivalued attributes
  - Create a second table that lists the primary key of the first table and one value of the multivalued attribute
    - Eg. Employee with many skills:
      Create the employee table and a second table called skills.
      A row in the skills table contains an employee id and a skill.

Multivalued Attributes

- Example: the employee table is to include a set of skills for each employee. The number of skills is varied and can grow.

- Solution 1: Create just one table that can list the skills:



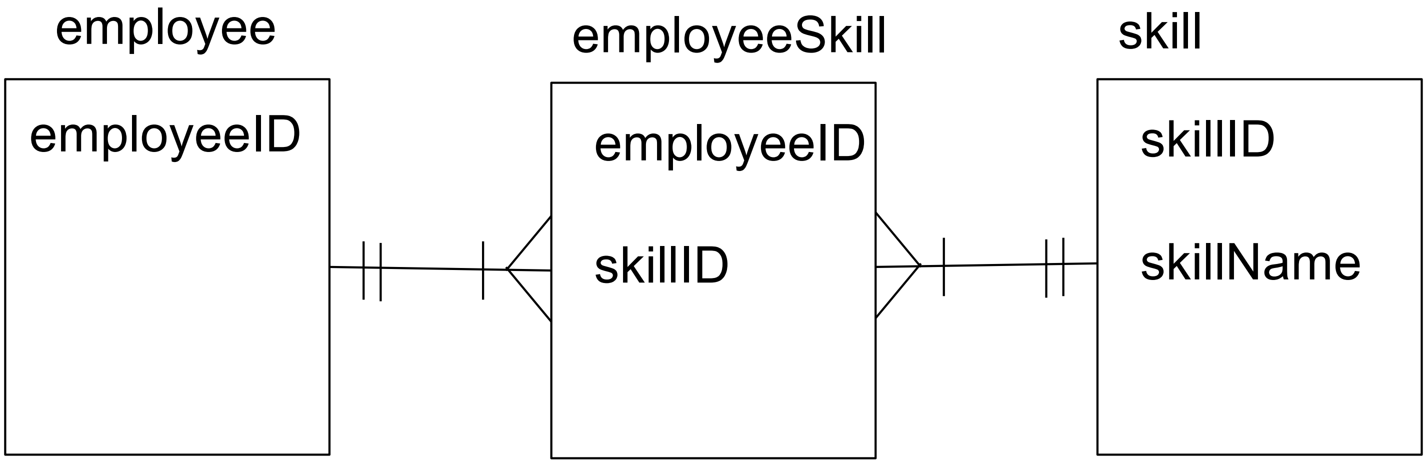
One employee can have many rows in the employeeSkill table.

No quick way to ensure that we're typing the skill names consistently.

Multivalued Attributes

- Example: the employee table is to include a set of skills for each employee. The number of skills is varied and can grow.

- Solution 2: Create a table of skills then join the two



Everyone with the same skill references the same ID in the skill table.

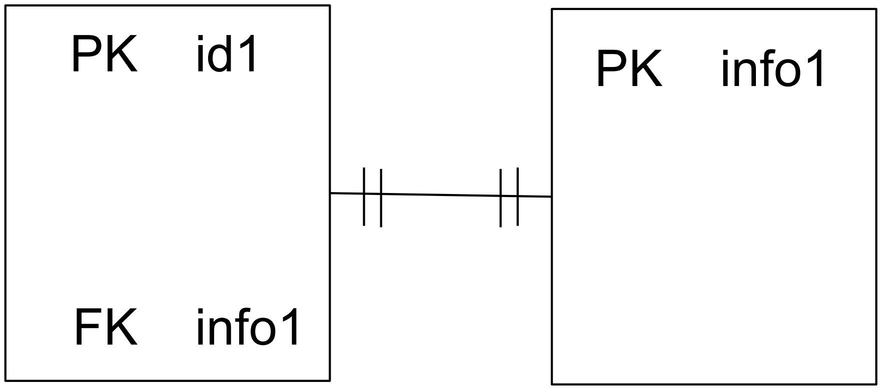
Map weak entities

- Recall that a weak entity is an entity that does not exist as an independent concept.
  - Eg. orderDetail in the lab database

- Create a table for the weak entity and include the primary key of the primary entity as a foreign key

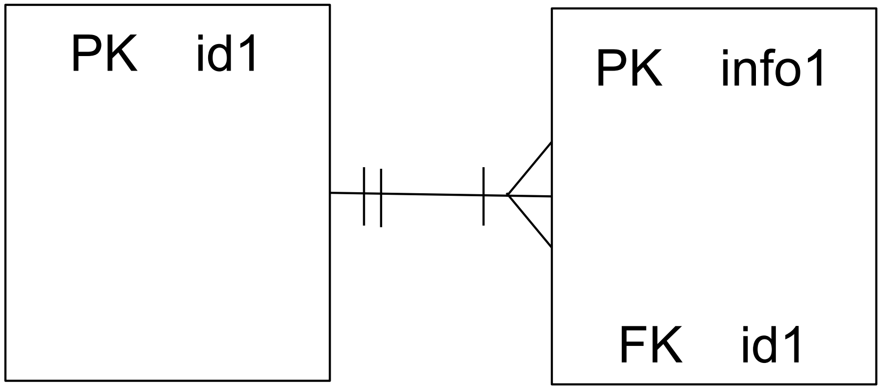
- Often use surrogate keys for weak entities

Map binary relations



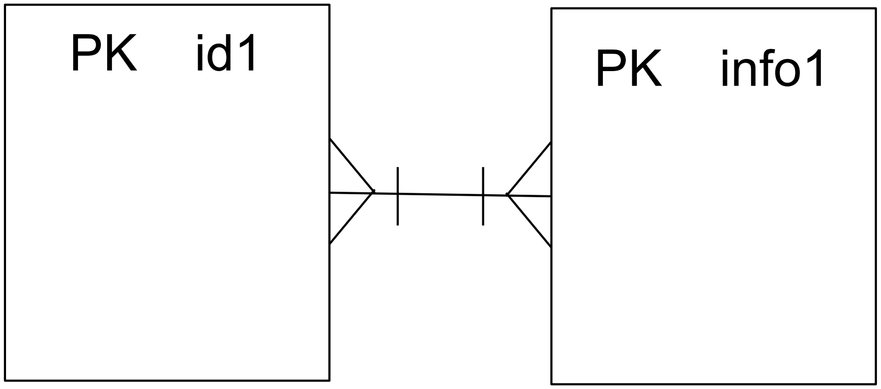
Include the primary key of one table as a foreign key in the other table.

Which table has the foreign key is context-dependent.



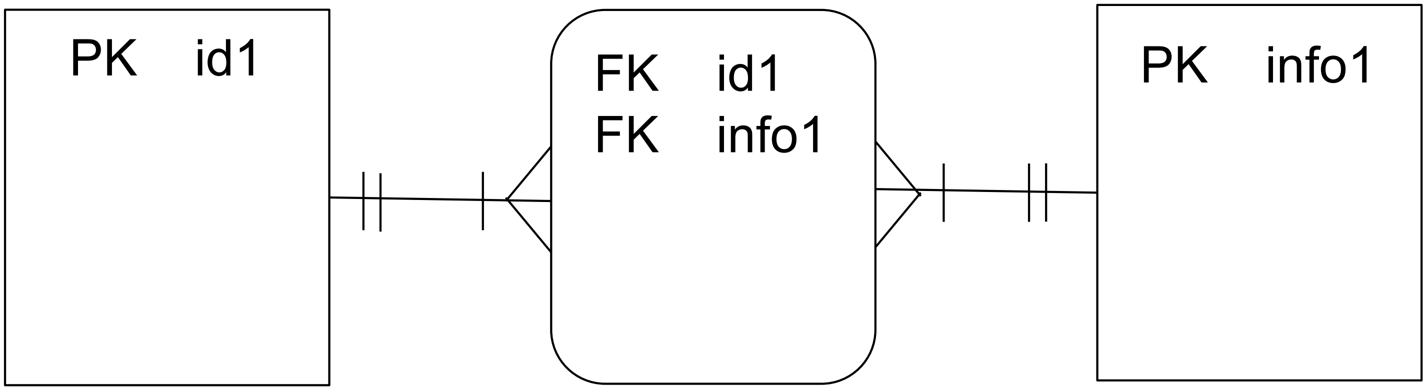
Include the primary key of the single element table as a foreign key in the multiple element table.

Map binary relations



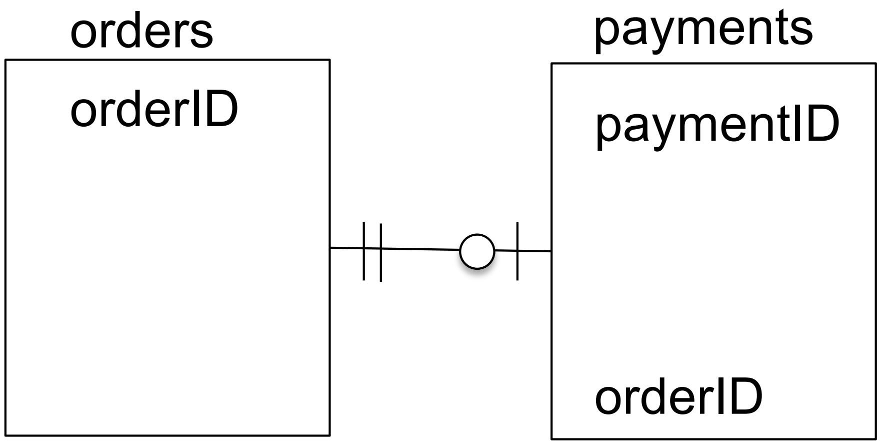
Create an intermediate table with the primary keys of both tables.

The key for this intermediate table is the set of both foreign keys.



Binary relations example

Business side: each order will have at most one payment and we don’t allow payments to cover more than one order



orders

orderID

payments

paymentID

orderID

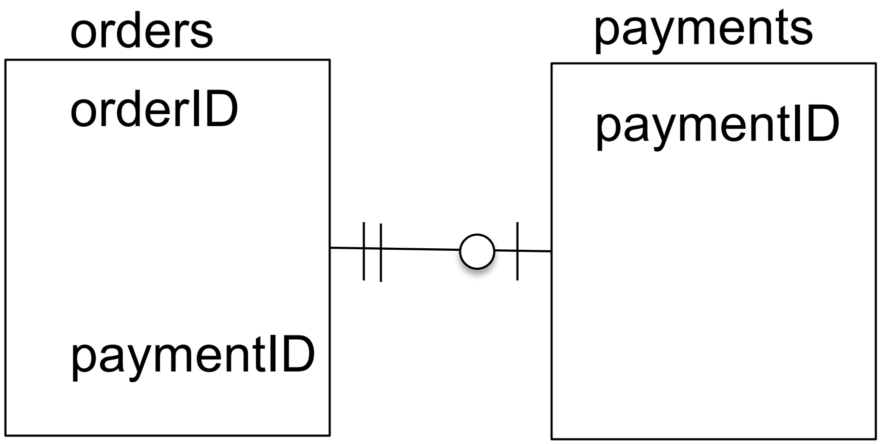
Note: 2 different ways to declare the primary keys. Both are ok.

```
create table orders (orderID int not null auto_increment primary key);

create table payments (paymentID int not null auto_increment,
      orderID int not null,
      primary key (paymentID),
      foreign key (orderId) references orders (orderID) );
```

Binary relations example – alternate solution

Business side: each order will have at most one payment and we don't allow payments to cover more than one order



Alternatively, have the paymentID in the order, but allow it to be NULL for the "no payments" option (less desirable solution, but still works).

```
create table payments (paymentID int not null auto_increment);

create table orders (orderID int not null auto_increment,
     paymentID int,
     primary key (orderID),
     foreign key (paymentID) references payments (paymentID) );
```