



# INFO20003 Database Systems

Dr Renata Borovica-Gajic

Lecture 05  
Modelling with MySQL Workbench

Week 3

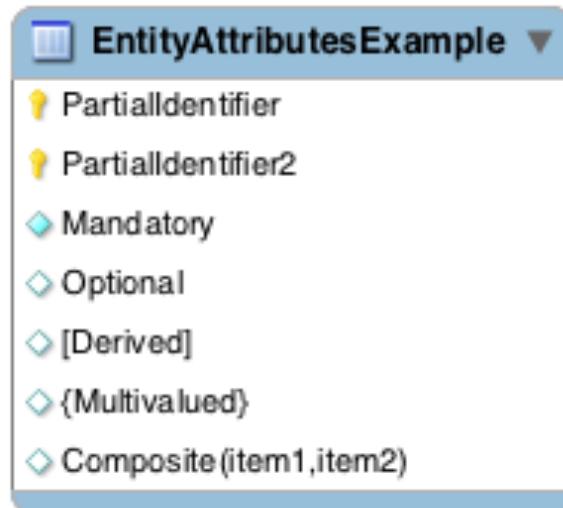
MELBOURNE

- Modelling with MySQL Workbench
- Recap & further design
  - Conceptual Design
  - Logical Design
  - Physical Design

- Entity



- Attributes



- Identifier or key:**

- Fully identifies an instance

- Partial Identifier:**

- Identifies an instance in conjunction with one or more partial identifiers

- Attributes types:**

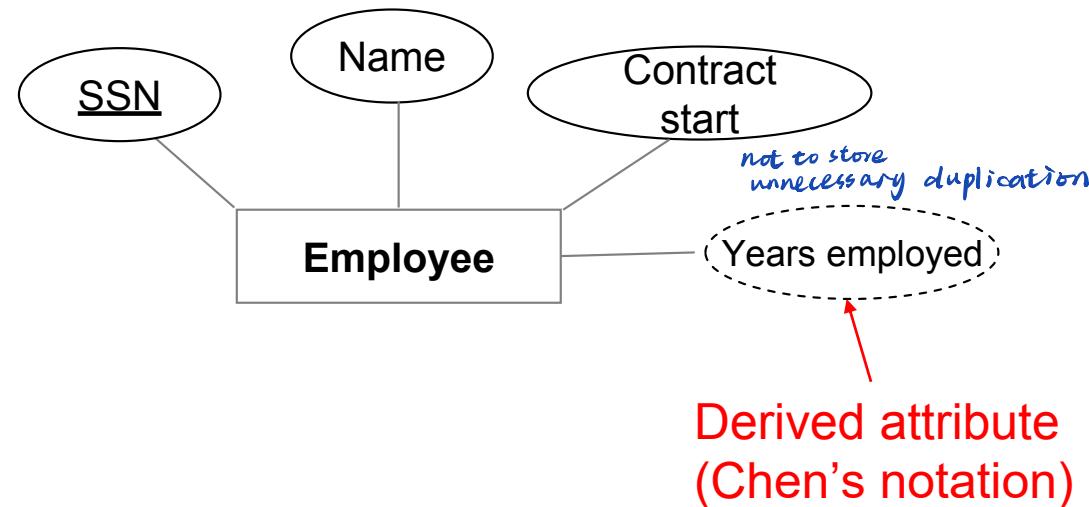
- Mandatory – NOT NULL (blue diamond)
- Optional - NULL (empty diamond)
- Derived []
  - [YearsEmployed]
- Multivalued {} *eg phone number*
  - {Skill}
- Composite () *has structure hidden inside*
  - Name (First, Middle, Last)

- Derived attributes imply that their values can be derived from some other attributes in the database. As a result, they do not need to be stored physically – they disappear at the physical design.



### Example:

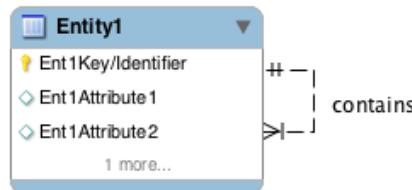
For employees we want to be able to show for how many years they have been employed.



MELBOURNE

- Relationship Degrees

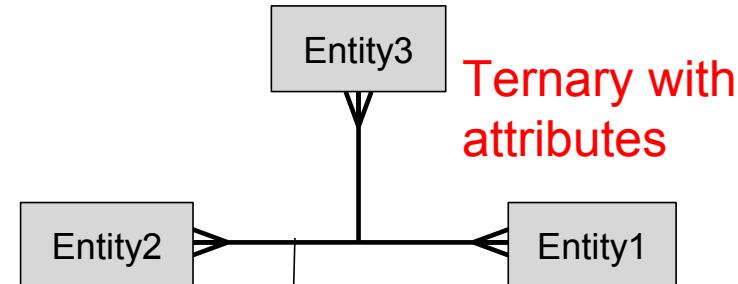
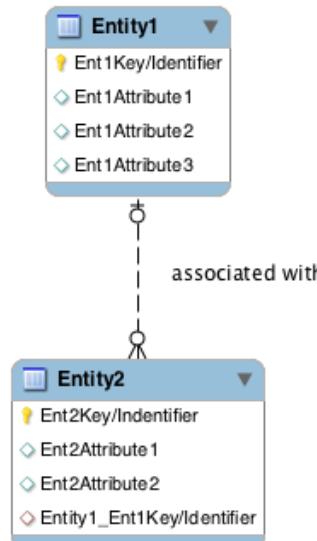
## Unary



## Ternary

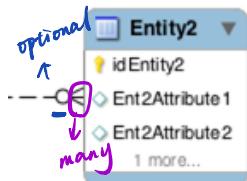


## Binary

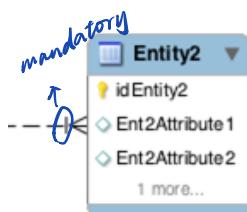


## Ternary with attributes

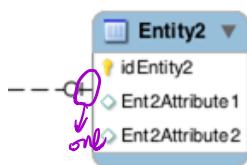
- Cardinality Constraints



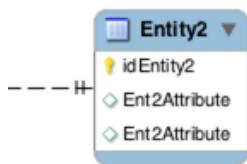
Optional Many  
Partial participation  
Without key constraint



Mandatory Many  
Total participation  
Without key constraint



Optional One  
Partial participation  
Key constraint



Mandatory One  
Total participation  
Key constraint

- Relationship Cardinality

- One to One

Each entity will have exactly 0 or 1 related entity

- One to Many

One of the entities will have 0, 1 or *more* related entities, the other will have 0 or 1.

- Many to Many

Each of the entities will have 0, 1 or *more* related entities

chen's Notation

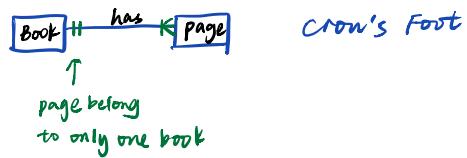
—	∅	m	optional many
—	1	m	Mandatory many
→	∅	1	optional one
→	1	1	mandatory one

Crow's Foot Notation

—	∅
—	1
—	of
—	11



chen's



Crow's Foot

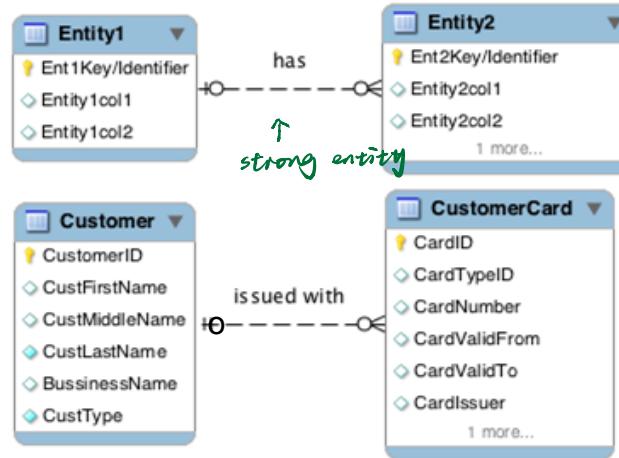




MELBOURNE

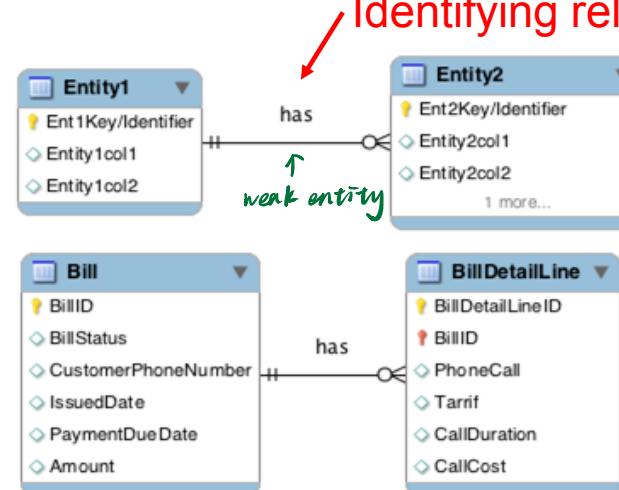
## Strong Entity:

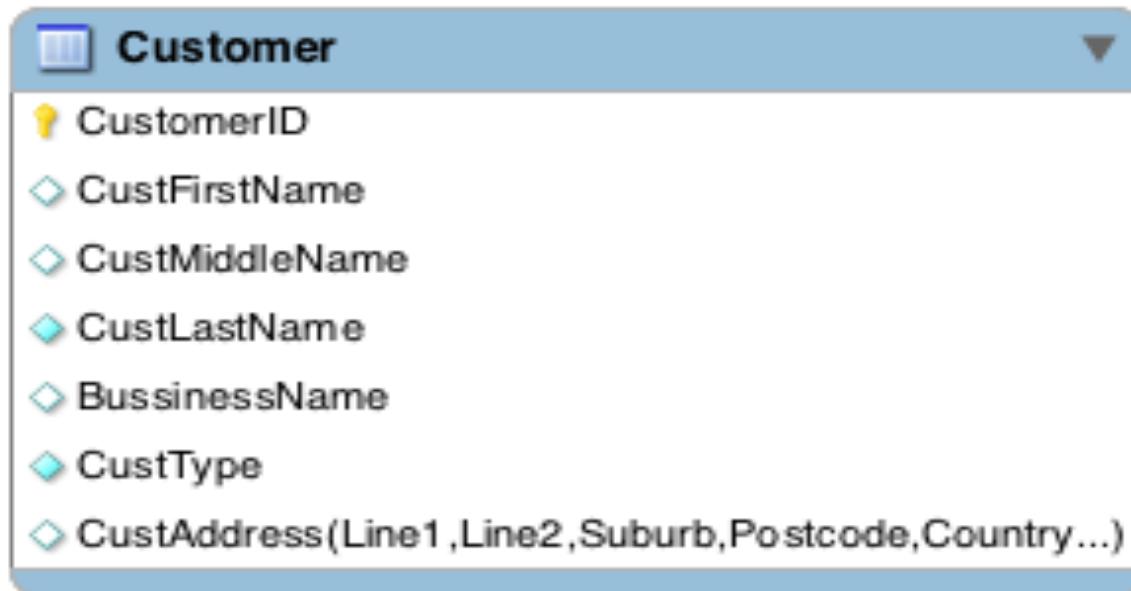
- Can exist by itself
- E.g. Customer Card & Customer



## Weak Entity

- Can't exist without the owner
- E.g. BillDetailLine





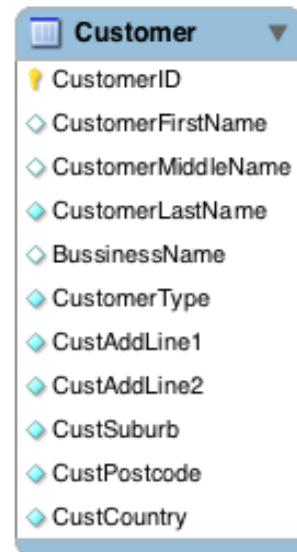


# Convert from Conceptual to Logical design



- Convert the ER into a logical (rel.) model
  - Customer(CustomerID,  
CustFirstName, CustMiddleName,  
CustLastName, BusinessName,  
CustType, CustAddLine1,  
CustAddLine2, CustSuburb,  
CustPostcode, CustCountry)

- Tasks checklist (from conceptual to logical):**
  - Flatten composite and multi-valued attributes
    - Multi-value attributes can become another table
  - Resolve many-many relationships
    - Create an associative entity
  - Resolve one-many relationships
    - Add foreign keys at crows foot end of relationships (on the many side in the case of crows foot)



- Generate attribute data types (with NULL/NOT NULL)

## Physical Design:

Customer	
mandatory	CustomerID INT
mandatory	CustomerFirstName VARCHAR(100)
mandatory	CustomerMiddleName VARCHAR(100)
mandatory	CustomerLastName VARCHAR(100)
mandatory	BussinessName VARCHAR(100)
	CustomerType CHAR(1)
	CustAddLine1 VARCHAR(100)
	CustAddLine2 VARCHAR(100)
	CustSuburb VARCHAR(60)
	CustPostcode CHAR(6)
	CustCountry VARCHAR(60)

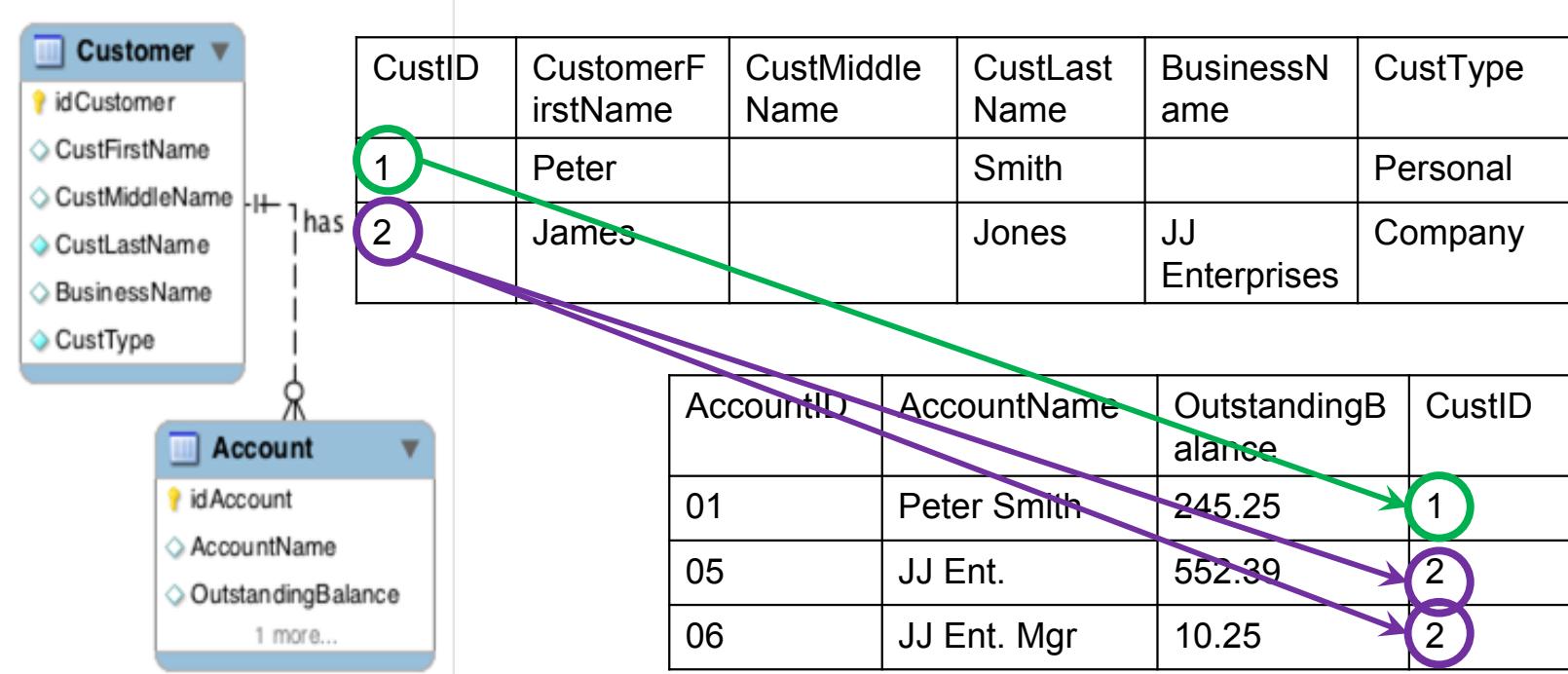
## Implementation:

```
CREATE TABLE Customer(
    CustomerID INT NOT NULL,
    CustFirstName VARCHAR(100),
    CustMiddleName VARCHAR(100),
    CustLastName VARCHAR(100) NOT NULL, mandatory
    BussinessName VARCHAR(100),
    CustType VARCHAR(1) NOT NULL,
    CustAddressLine1 VARCHAR(100) NOT NULL,
    CustAddressLine2 VARCHAR(100) NOT NULL,
    CustSuburb VARCHAR(60) NOT NULL,
    CustPostcode CHAR(6) NOT NULL,
    CustCountry VARCHAR(60) NOT NULL,
    PRIMARY KEY (CustomerID));
```



MELBOURNE

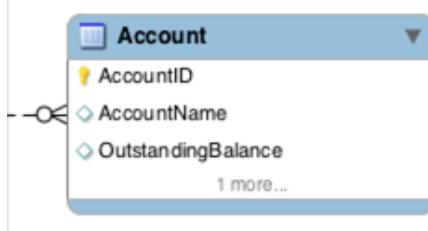
- A customer can have a number of Accounts
- The tables are linked through a foreign key





MELBOURNE

## Conceptual Design:



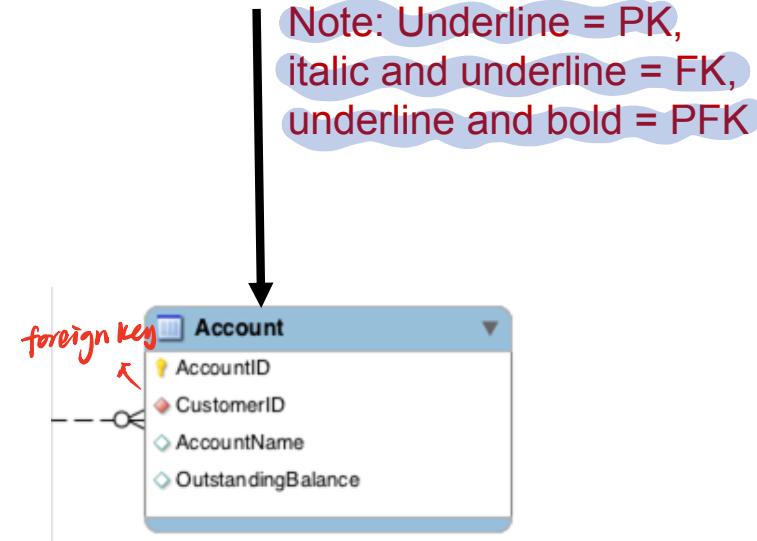
### Tasks checklist:

1. Flatten composite and multi-valued attributes **X**
2. Resolve many-many relationships **X**
3. Resolve one-many relationships
  - See FK1 – CustomerID
  - Every row in the account table must have a CustomerID from Customer (referential integrity)

## Logical Design:

Account(AccountID,  
AccountName,  
OutstandingBalance,  
CustomerID)

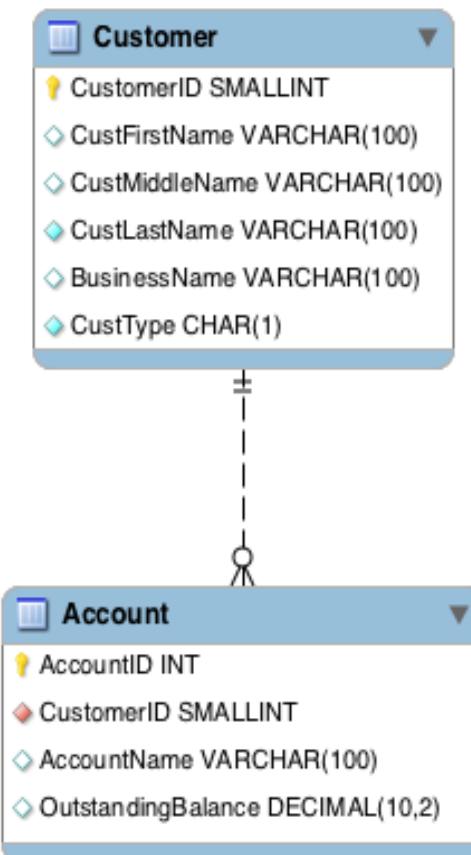
Note: Underline = PK,  
italic and underline = FK,  
underline and bold = PFK





MELBOURNE

## Physical design:



## Implementation:

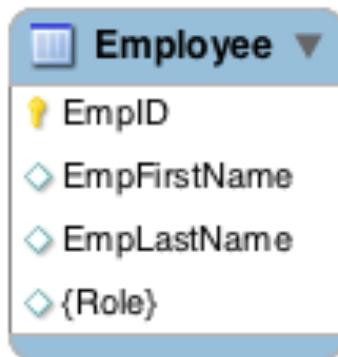
```
CREATE TABLE Account (
    AccountID int auto_increment,
    AccountName varchar(100) NOT NULL,
    OutstandingBalance DECIMAL(10,2) NOT NULL,
    CustomerID smallint NOT NULL,
    PRIMARY KEY (AccountID),
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)
        ON DELETE RESTRICT
        ON UPDATE CASCADE
) ENGINE=InnoDB;
```

*→ can't delete the customer if customer gets update, the account will also get update*

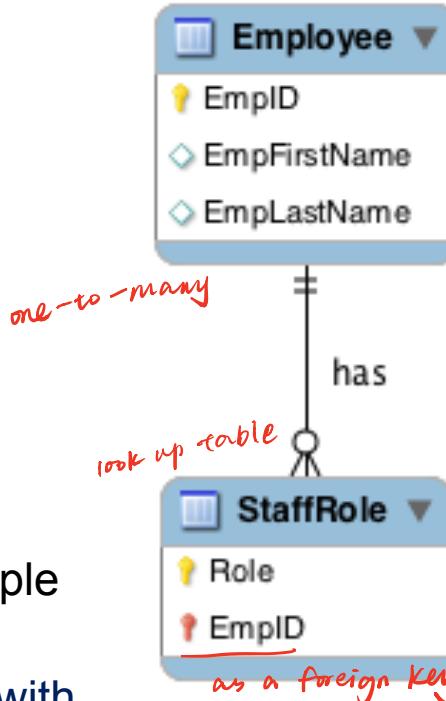


## Dealing with Multi-Valued Attributes: Approach 2

## Conceptual Design:



## Logical Design:

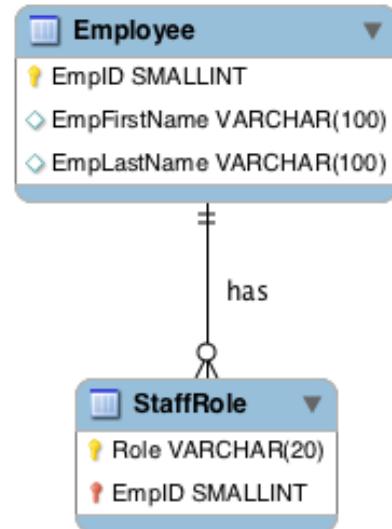


StaffRole is an example of a weak entity

- We show this with a *solid* line in Workbench

## Physical Design:

① usually flatten it  
if just finite attributes  
② lookup table (add data type)

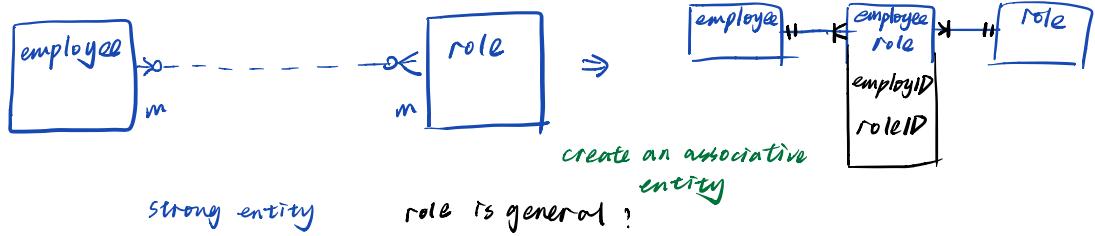


If staff have only 2-3 roles, you may decide to have these within the Employee table at physical design to save on "JOIN" time

1) flattening

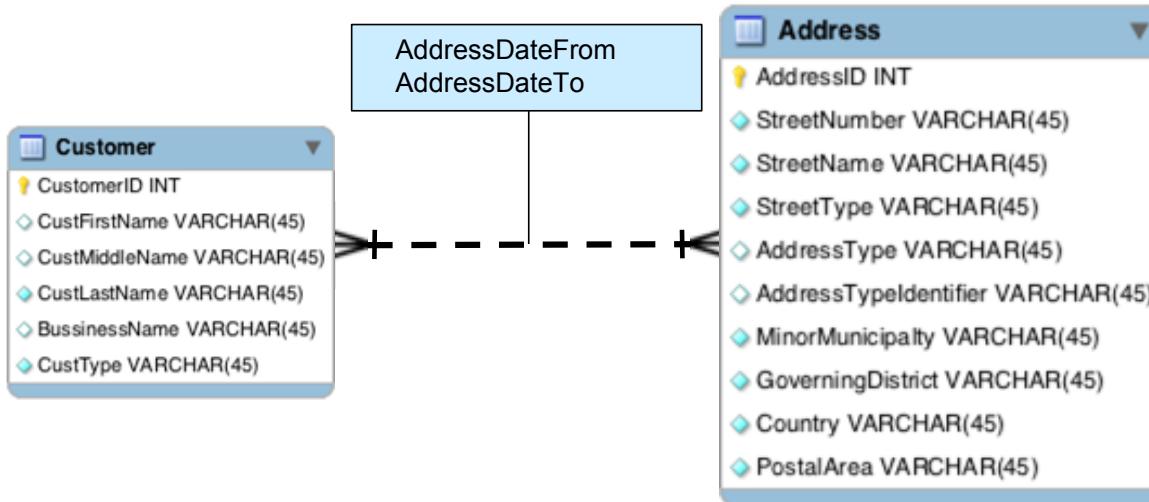
2) 1 ... m      look up table

3) m ... m





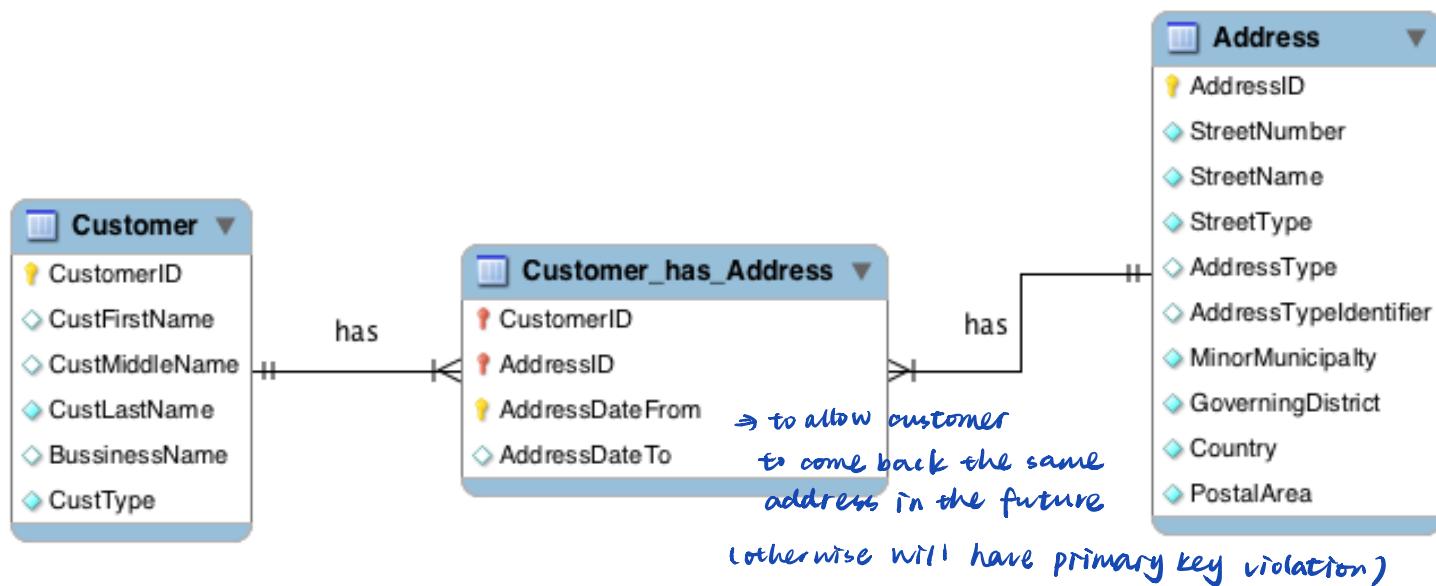
- How do we deal with customer addresses?
  - If customers can change addresses
    - AND imagine that we *need* to store a history of addresses for customers.
  - At the conceptual level it looks like this:





MELBOURNE

- When converting the conceptual to logical diagram we create an **Associative Entity** between the other 2 entities

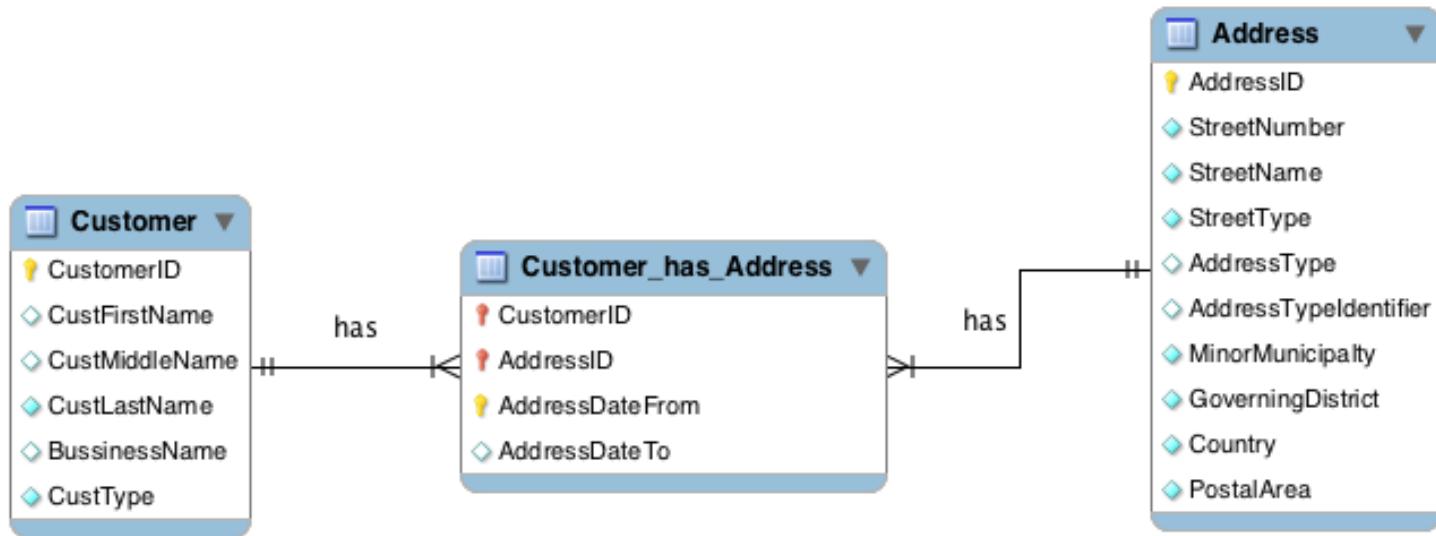


Note: AddressDateFrom/To are descriptive attributes of the relationship  
They go into the associative entity for M-M

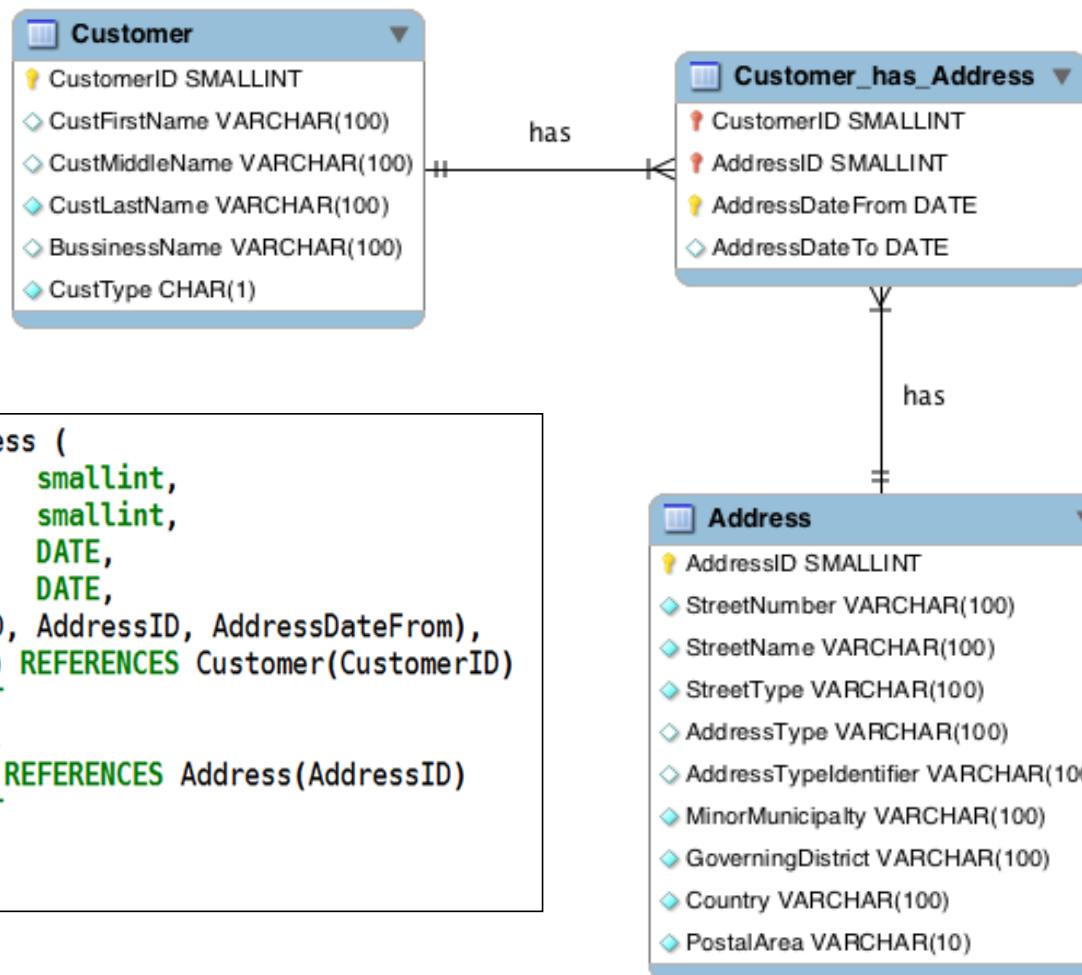
MELBOURNE

- Customer(CustomerID, CustFirstName, CustMiddleName, CustLastName, BusinessName, CustType)
- Address(AddressID, StreetNumber, StreetName, StreetType, AddressType, AddressTypeID, MinorMunicipality, MajorMunicipality, GoverningDistrict, Country, PostalArea)
- Customer\_Has\_Address(**CustomerID**, **AddressID**, AddressDateFrom, AddressDateTo)

Note: Underline = PK, italic and underline = FK, underline and bold = PFK

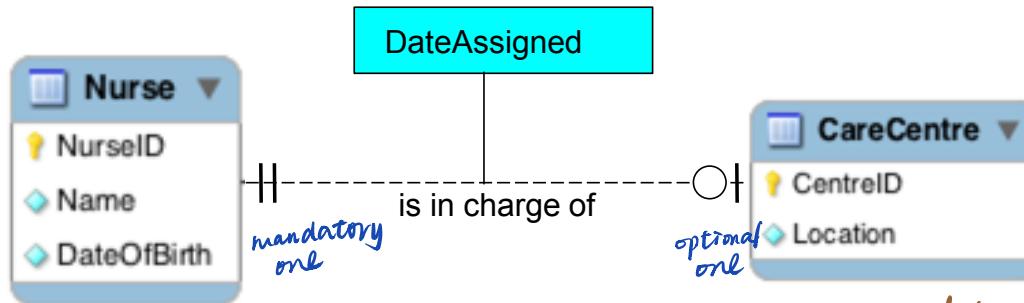


# Many to Many - Physical Model & Implementation



MELBOURNE

- Rule: Move the key from the *one* side to the other side



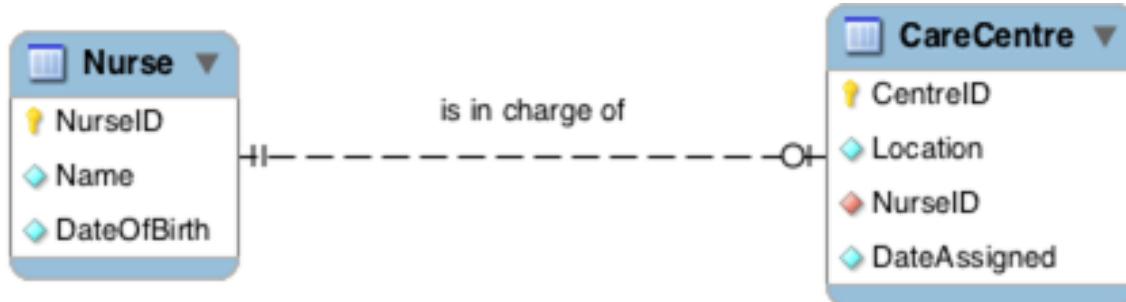
*sides*  
 if two *mandatory one* sides,  
 just move the key to the  
 side arbitrarily.

- But we have 2 “one” sides. Which one?
- Need to decide whether to put the foreign key inside Nurse or CareCentre (in which case you would have the Date\_Assigned in the same location)
  - Where would the least NULL values be?
  - The rule is the OPTIONAL side of the relationship gets the foreign key  
*have least NULL values*

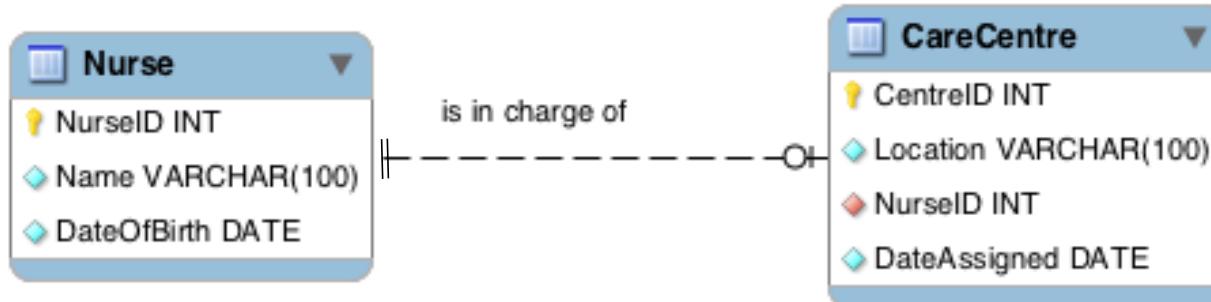


- **Logical Design:**

- Nurse(NurseID, Name, DateOfBirth)
- CareCentre(CentreID, Location, NurseID, DateAssigned)



- **Physical Design:**



MELBOURNE

- **One-to-Many**
  - Primary key on the one side becomes a foreign key on the many side (in the case of Crow's foot)
- **Many-to-Many**
  - Create an Associative Entity (a new relation) with the primary keys of the two entities it relates to as the combined primary key
- **One-to-One**
  - Need to decide where to put the foreign key
  - The primary key on the mandatory side becomes a foreign key on the optional side
  - If two optional or two mandatory, pick one arbitrarily

for Chen's notation

key from many side to one side

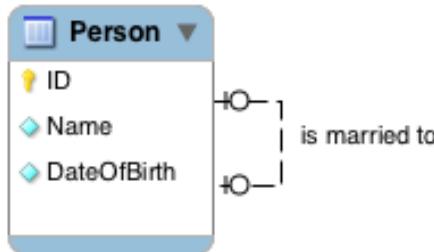
MELBOURNE

- Operate in the same way as binary relationships
  - **One-to-One**
    - Put a Foreign key in the relation
  - **One-to-Many**
    - Put a Foreign key in the relation
  - **Many-to-Many**
    - Generate an Associative Entity
    - Put two Foreign keys in the Associative Entity
      - Need 2 different names for the Foreign keys
      - Both Foreign keys become the *combined* key of the Associative Entity

move key from one side to many side

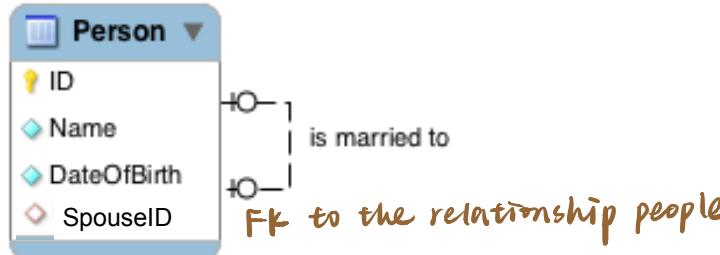
MELBOURNE

## Conceptual Design:



## Logical Design:

- Person (ID, Name, DateOfBirth, SpouseID)



## Implementation:

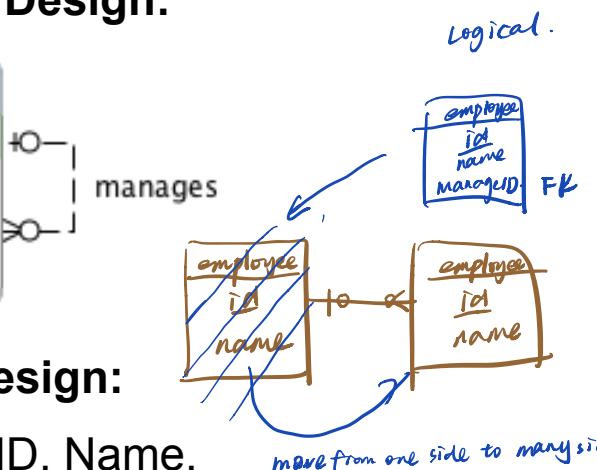
```

CREATE TABLE Person (
    ID INT NOT NULL,
    Name VARCHAR(100) NOT NULL,
    DateOfBirth DATE NOT NULL,
    SpouseID INT,
    PRIMARY KEY (ID),
    FOREIGN KEY (SpouseID)
    REFERENCES Person (ID) → refer to the
    ON DELETE RESTRICT
    ON UPDATE CASCADE);
  
```

ID	Name	DOB	SpouseID
1	Ann	1969-06-12	3
2	Fred	1971-05-09	NULL
3	Chon	1982-02-10	1
4	Nancy	1991-01-01	NULL



## Conceptual Design:

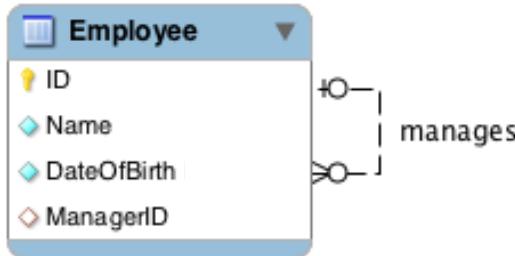


## Implementation:

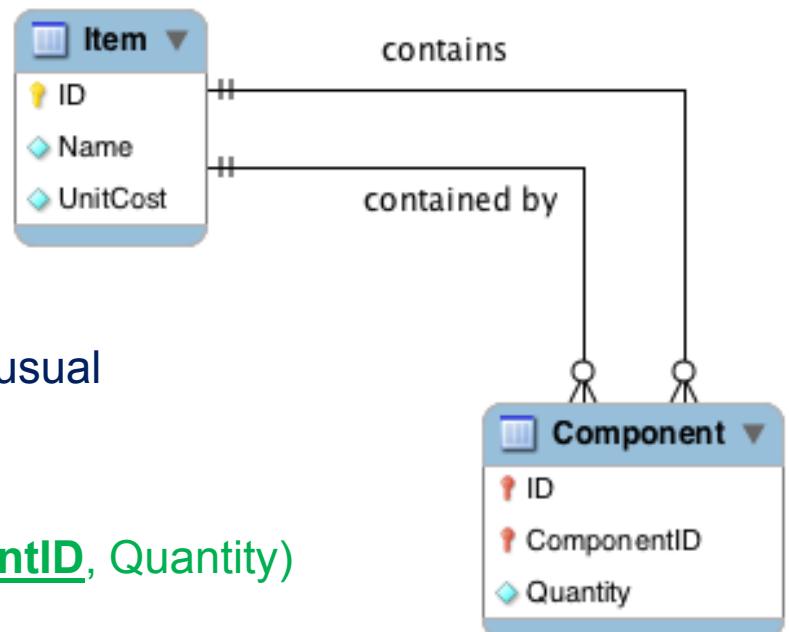
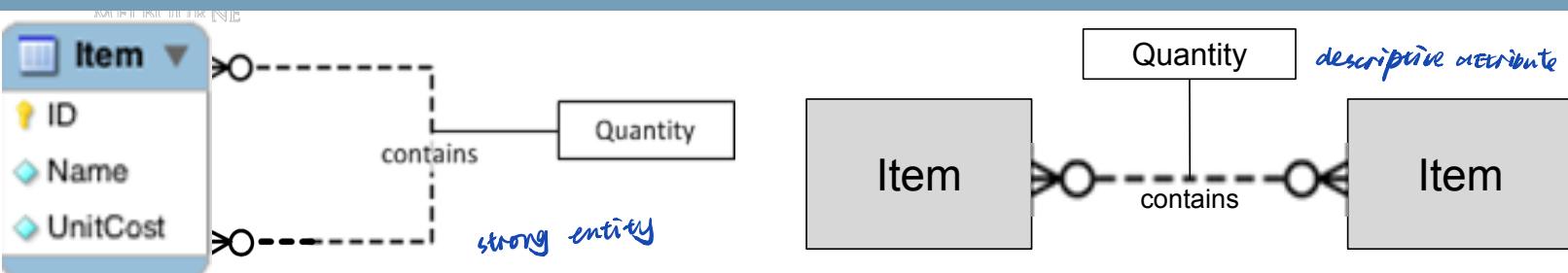
```
CREATE TABLE Employee(  
    ID smallint NOT NULL,  
    Name VARCHAR(100) NOT NULL,  
    DateOfBirth DATE NOT NULL,  
    ManagerID smallint,  
    PRIMARY KEY (ID),  
    FOREIGN KEY (ManagerID)  
    REFERENCES Employee(ID)  
    ON DELETE RESTRICT  
    ON UPDATE CASCADE);
```

## Logical Design:

- Employee (ID, Name, DateOfBirth, ManagerID)



ID	Name	DOB	MngrID
1	Ann	1969-06-12	NULL
2	Fred	1971-05-09	1
3	Chon	1982-02-10	1
4	Nancy	1991-01-01	1



- **Logical Design:**

- Create Associative Entity like usual
- Generate logical model
  - **Item(ID, Name, UnitCost)**
  - **Component(ID, ComponentID, Quantity)**

- Implementation

```
CREATE TABLE Part (
```

```
    ID          smallint,  
    Name        VARCHAR(100)  NOT NULL,  
    UnitCost    DECIMAL(6,2)  NOT NULL,  
    PRIMARY KEY (ID)  
) ENGINE=InnoDB;
```

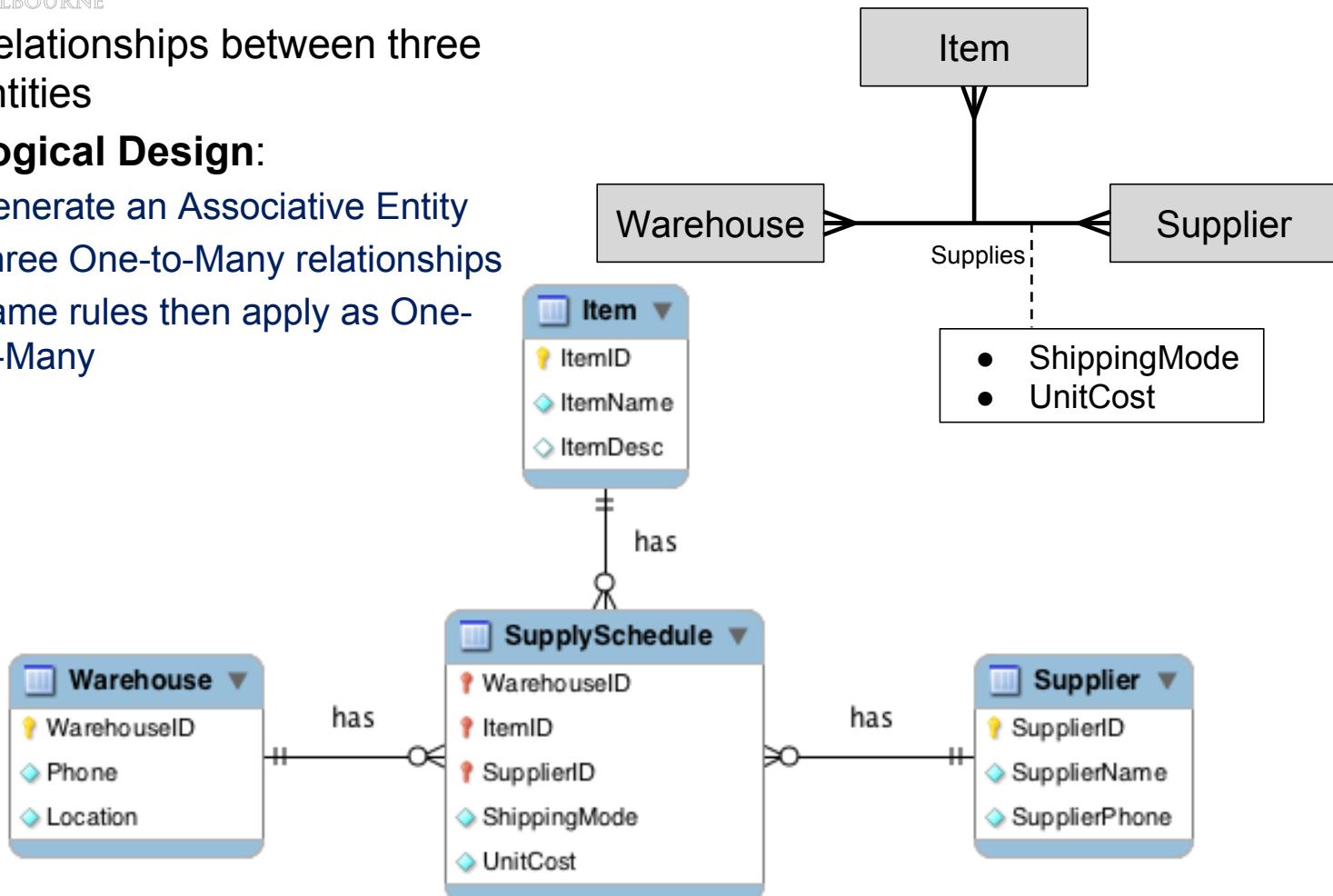
```
CREATE TABLE Component (
```

```
    ID          smallint,  
    ComponentID smallint,  
    Quantity    smallint  NOT NULL,  
    PRIMARY KEY (ID, ComponentID),  
    FOREIGN KEY (ID) REFERENCES Part(ID)  
        ON DELETE RESTRICT  
        ON UPDATE CASCADE,  
    FOREIGN KEY (ComponentID) REFERENCES Part(ID)  
        ON DELETE RESTRICT  
        ON UPDATE CASCADE  
) ENGINE=InnoDB;
```



MELBOURNE

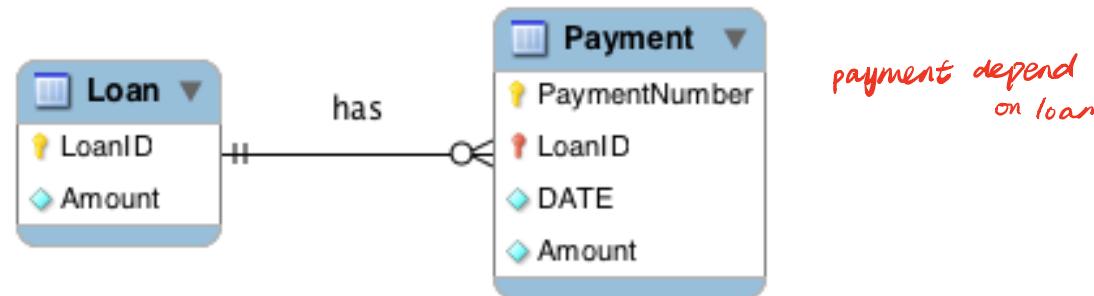
- Relationships between three entities
- Logical Design:**
  - Generate an Associative Entity
  - Three One-to-Many relationships
  - Same rules then apply as One-to-Many





MELBOURNE

- How to map an Identifying relationship
  - Map it the same way: Foreign Key goes into the relationship at the crow's foot end.
  - Only Difference is: The Foreign Key becomes part of the Primary Key



- Logical Design:
  - **Loan(LoanID, Amount)**
  - **Payment(PaymentNumber, LoanID, Date, Amount)**
- Physical Design: as per normal one-to-many

**Concept**    **Chen's not. Crow's foot not.**

Entity		
Weak Entity		
Attribute		
Multivalued A.		*
Composite A.		*
Derived A.		*
Key A.		*
Weak (or Partial) Key A.		*
Relationship		
Weak (Identifying) Relationship		

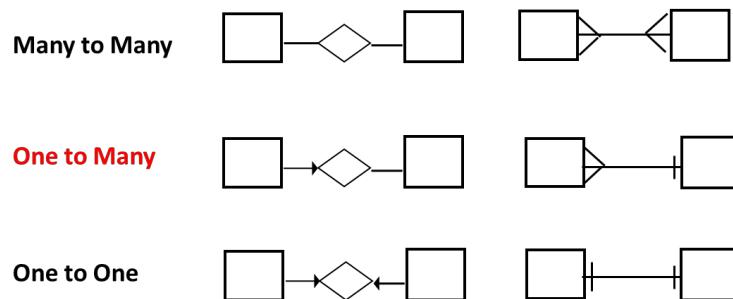
**Relationship cardinalities and constraints**

Chen's notation    Crow's foot notation

Optional Many 0..m		
Mandatory Many 1..m		
Optional One 0..1		
Mandatory One 1..1		

**BINARY Relationship Cardinalities**

Here we just looked at cardinalities and omitted participation constraints (optional/mandatory) for clarity



MELBOURNE

- Need to be able to draw conceptual, logical and physical diagrams
  - Assignment 1: Conceptual Chen's pen and paper, Physical Crow's foot with MySQL Workbench
  - Create table SQL statements

## MELBOURNE

- Hands on Modelling
- Please read the case study prior to the lecture:
  - LMS/Week 3 Medicare study