Dr Greg Wadley

INFO 90002 Database Systems & Information Modelling

Week 04

Data Modelling 2

Today's Session…

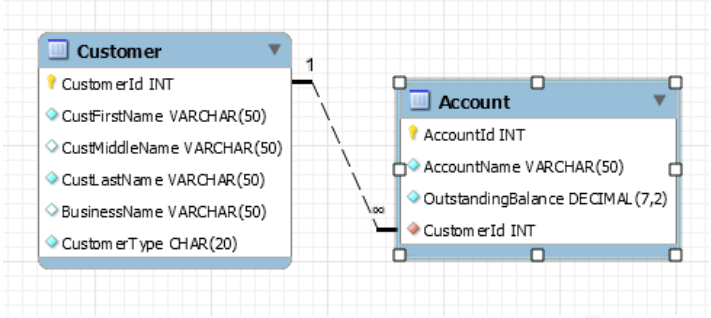
- More relationship types
  - Many to Many
    - Associative entity
  - One to One
  - Recursive / Unary relationships
    - One-to-one, One-to-many, Many-to-Many
  - Multiple One to Many Relationships
    - between the same pair of entities
- Data Modelling
  - Ternary relationships
    - 3 tables are involved

© University of Melbourne 2018

Recap: one-to-many relationships

- Data are spread across 2 tables
- Inner join = Join rows where FK value = PK value

CustID	CustFirstName	CustMiddleName	CustLastName	BusinessName	CustType
1	Peter		Smith		Personal
2	James		Jones	JJ Enterprises	Company



AccountID	AccountName	OutstandingBalance	CustID
01	Peter Smith	245.25	1
05	JJ Ent.	552.39	2
06	JJ Ent. Mgr	10.25	2

```
SELECT *
FROM Customer INNER JOIN Account
ON Customer.Customerid = Account.Customerid;
```

-3-

CustomerId	CustFirstName	CustMiddleName	CustLastName	BusinessName	CustomerType	AccountId	AccountName	OutstandingBalance	CustomerId
1	Peter	NULL	Smith	NULL	Personal	1	Peter Smith	245.25	1
2	James	NULL	Jones	JJ Enterprises	Company	5	JJ Ent.	552.39	2
2	James	NULL	Jones	JJ Enterprises	Company	6	JJ Ent. Mgr	10.25	2

© University of Melbourne 2018

Data Integrity Constraints

- Domain Integrity
  - Valid values and domain
    - selection of data type constrains possible data values
  - Default value
    - takes this value if no explicit value is given on Insert
  - Null value control
    - allows or prohibits empty fields
  - Check constraint
    - limits range of allowable values (not available in MySQL)
- Entity Integrity Constraints
  - Primary key cannot be null
  - No component of a composite key can be null
  - Primary key must be unique

© University of Melbourne 2018

Referential Integrity

- Each non-null FK value must match a PK value
  - Rules for update and delete (SQL CREATE statement)
    - RESTRICT
      - Don't allow deletes or updates of the parent table if related rows exist in the child table
    - CASCADE
      - Automatically delete/update the child table if related rows are deleted/updated in the parent table
    - SET NULL
      - Set the foreign key to NULL in the child table if deleting/updating the key in parent table

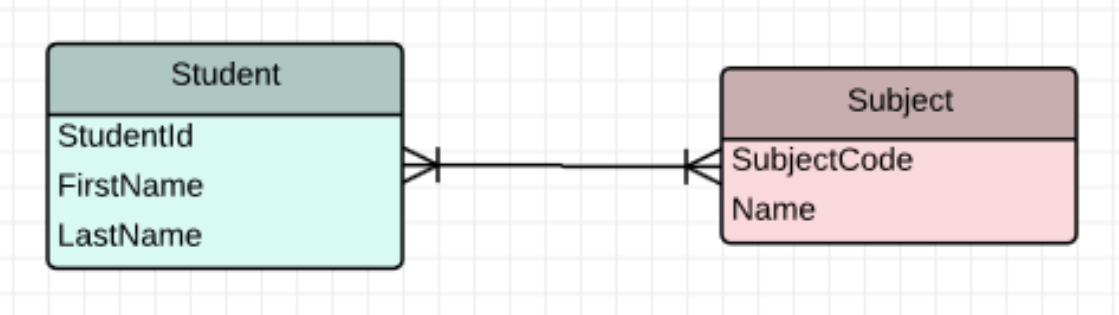
```
CONSTRAINT `fk_StudentSubject_Student`
  FOREIGN KEY (`StudentId`)
  REFERENCES `Student` (`StudentId`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION,
CONSTRAINT `fk_StudentSubject_Subject1`
  FOREIGN KEY (`SubjectCode`)
  REFERENCES `Subject` (`SubjectCode`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION)
```


© University of Melbourne 2018

Many-to-Many Relationships

Many to Many relationships

- Example: we need to design a Student Records database
- Each student will take more than one subject,
  and each subject will be taken by more than one student
- Where do we record who took what subject and their result?



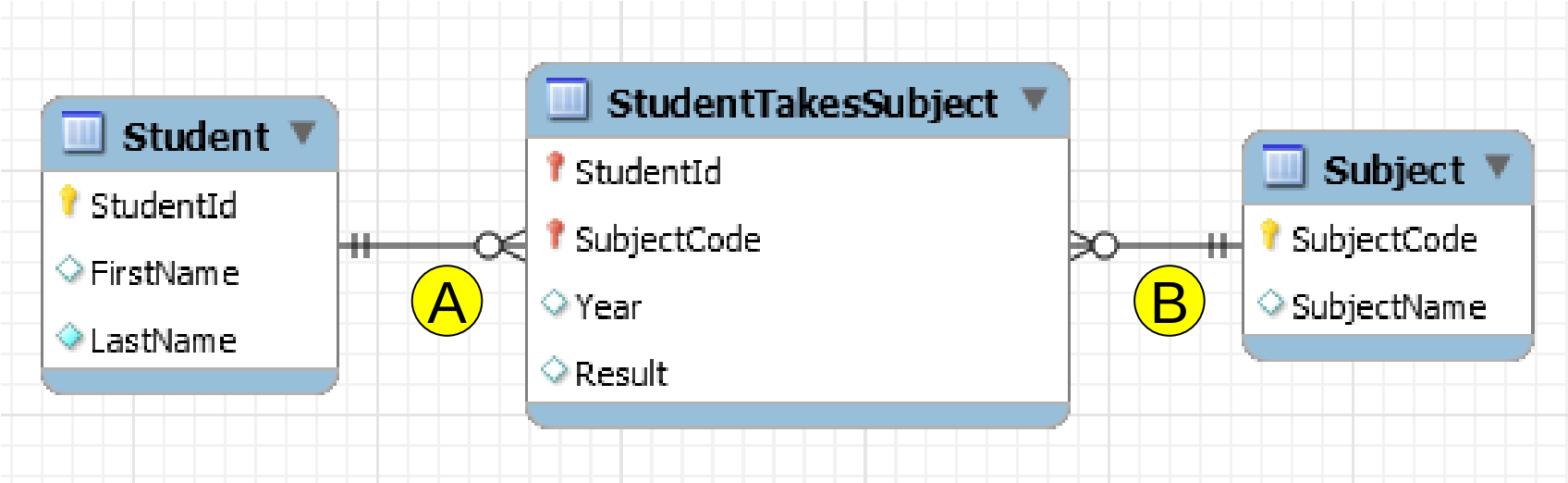
StudentId	FirstName	LastName
11111	John	Lennon
22222	Paul	McCartney
33333	George	Harrison

SubjCode	Name
INFO90002	Database
ISYS90026	Fundamentals
ISYS90081	Organisational

© University of Melbourne 2018

Problems modelling Many-Many…

- Relational database doesn't directly support M-M…
  - so we create an Associative Entity between the other 2 entities (when converting Conceptual to Logical model)
  - each of these 2 relationships is like any 1-M relationship

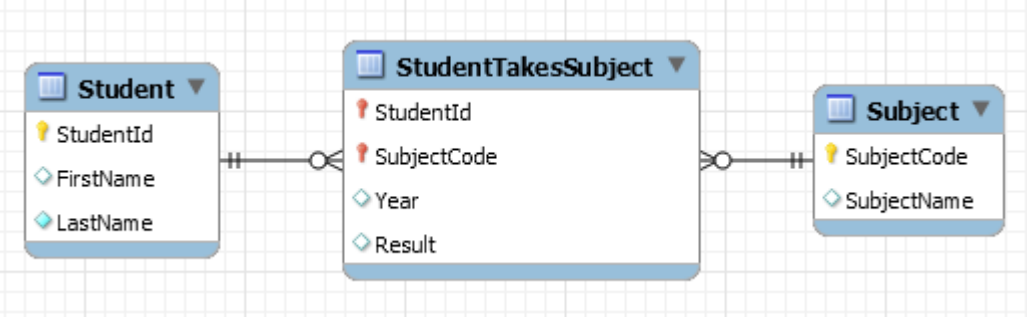


- We can add attributes to the associative entity to record when the student took the subject and the result they got.
- Associate Entities are also called 'Join Tables' and many other names, see https://en.wikipedia.org/wiki/Junction_table

© University of Melbourne 2018

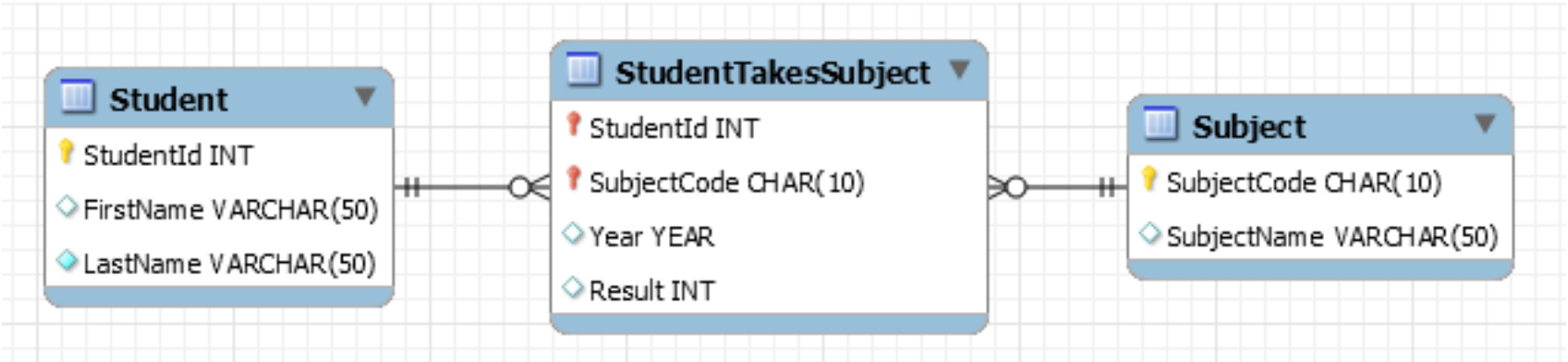
Associative Entities

- When to create
  - when going from Conceptual to Logical phase of design
  - to implement a Many-to-Many relationship
  - to implement a Ternary relationship
- The associative entity
  - has an independent meaning
  - has a unique identifier, usually a combination of FKs
  - may have attributes other than the FKs
  - may participate in other relationships



© University of Melbourne 2018

Many-Many Physical Model



- Choose data types
- Example decisions:
  - are results integers or floating point?
  - are StudentIds number or strings?
  - how long are people's names?

© University of Melbourne 2018

Many-Many CREATE statements

- Order of creation is important!
  - so is order of deletion…
- Create tables *without* foreign keys first
  - *drop* tables without foreign keys *last*

```
-- -----------------------------------------------------
-- Table `Student`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Student` (
  `StudentId` INT NOT NULL,
  `FirstName` VARCHAR(50) NULL,
  `LastName` VARCHAR(50) NULL,
  PRIMARY KEY (`StudentId`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Subject`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Subject` (
  `SubjectCode` CHAR(10) NOT NULL,
  `SubjectName` VARCHAR(50) NULL,
  PRIMARY KEY (`SubjectCode`))
ENGINE = InnoDB;
```


© University of Melbourne 2018

Many-Many CREATE statements

- Order of creation is important!
  - so is order of deletion…
- Create tables *with* foreign keys last
  - *drop* tables with foreign keys *first*

```
-- -----------------------------------------------------
-- Table `StudentTakesSubject`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `StudentTakesSubject` (
  `StudentId` INT NOT NULL,
  `SubjectCode` CHAR(10) NOT NULL,
  `Year` YEAR NULL,
  `Result` INT NULL,
  PRIMARY KEY (`StudentId`, `SubjectCode`),
  INDEX `fk_StudentSubject_Subject1_idx` (`SubjectCode` ASC),
  CONSTRAINT `fk_StudentSubject_Student`
    FOREIGN KEY (`StudentId`)
    REFERENCES `Student` (`StudentId`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_StudentSubject_Subject1`
    FOREIGN KEY (`SubjectCode`)
    REFERENCES `Subject` (`SubjectCode`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```


© University of Melbourne 2018

Adding data to a M-M relationship

- Insert into the join table *last*

```
-- ----------------------------------------------------
-- Data for table `Student`
-- ----------------------------------------------------
INSERT INTO `Student` VALUES (11111, 'John', 'Lennon');
INSERT INTO `Student` VALUES (22222, 'Paul', 'McCartney');
INSERT INTO `Student` VALUES (33333, 'George', 'Harrison');
INSERT INTO `Student` VALUES (44444, 'Ringo', 'Starr');
-- ----------------------------------------------------
-- Data for table `Subject`
-- ----------------------------------------------------
INSERT INTO `Subject` VALUES ('INFO90002', 'Database Systems and Information Modelling');
INSERT INTO `Subject` VALUES ('ISYS90026', 'Fundamentals of Information Systems');
INSERT INTO `Subject` VALUES ('ISYS90081', 'Organisational Processes');
INSERT INTO `Subject` VALUES ('ISYS90048', 'Managing ICT Infrastructure');
INSERT INTO `Subject` VALUES ('ISYS90045', 'Professional ICT Consulting');
-- ----------------------------------------------------
-- Data for table `StudentTakesSubject`
-- ----------------------------------------------------
INSERT INTO `StudentTakesSubject` VALUES (11111, 'INFO90002', 2014, 85);
INSERT INTO `StudentTakesSubject` VALUES (11111, 'ISYS90026', 2015, 77);
INSERT INTO `StudentTakesSubject` VALUES (22222, 'INFO90002', 2014, 90);
```


© University of Melbourne 2018

How to read complete student results

- Three table join

```
1 • SELECT *
2   FROM Student NATURAL JOIN StudentTakesSubject
3   NATURAL JOIN Subject;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

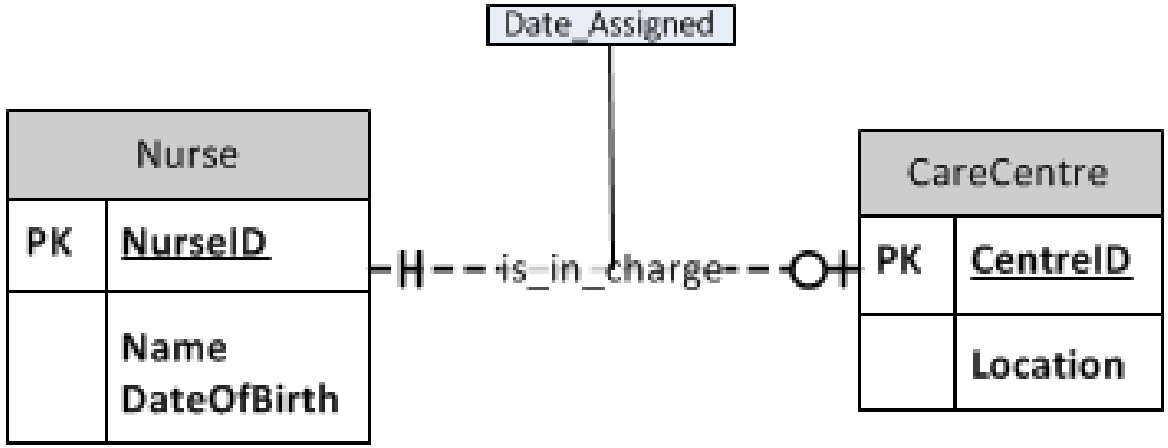
SubjectCode	StudentId	FirstName	LastName	Year	Result	SubjectName
INFO90002	11111	John	Lennon	2014	85	Database Systems and...
ISYS90026	11111	John	Lennon	2015	77	Fundamentals of Info...
INFO90002	22222	Paul	McCartney	2014	90	Database Systems and...

© University of Melbourne 2018

One-to-One Relationships

Binary One-One Relationship

- Given this example… How do we implement it…



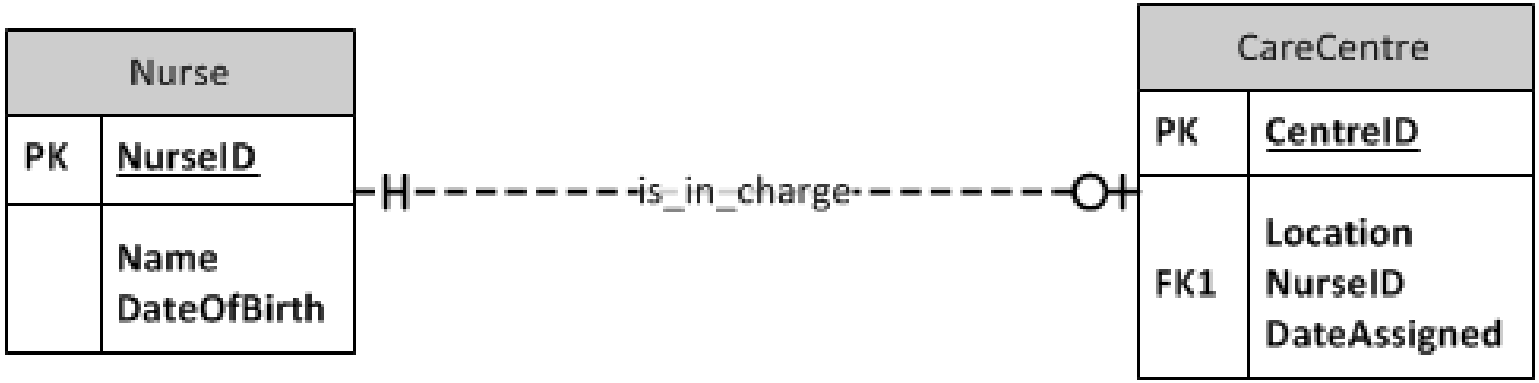
- Note: Date_assigned is an attribute of the relationship

- 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

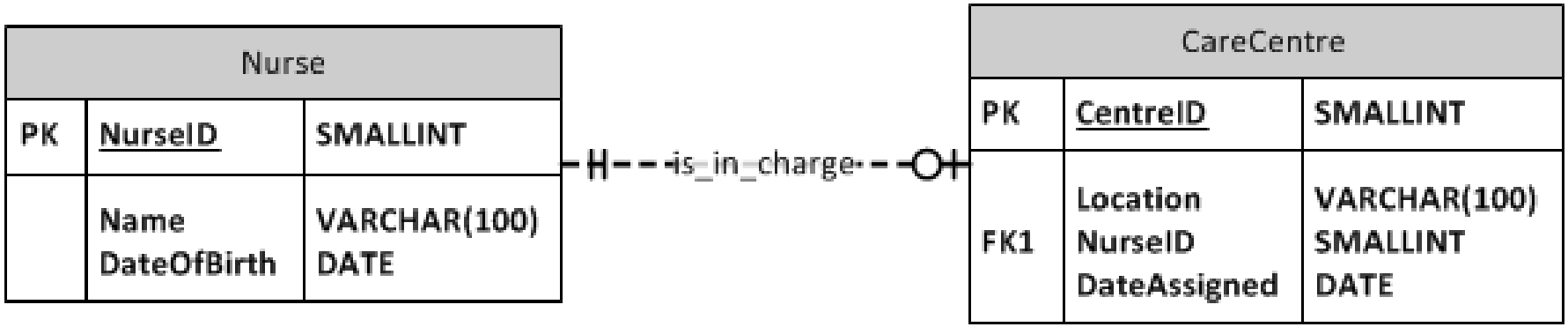
© University of Melbourne 2018

Binary One-One Relationship – Logical and Physical

- Logical
  - Nurse = (NurseID, Name, DateOfBirth)
  - CareCentre = (CentreID, Loction, NurseID, DateAssigned)



- Physical



© University of Melbourne 2018

1-1 Implementation in SQL

```
CREATE TABLE Nurse (
  NurseID                 smallint,
  Name                    varchar(100)     NOT NULL,
  DateOfBirth             varchar(100)     NOT NULL,
  PRIMARY KEY  (NurseID)
) ENGINE=InnoDB;


CREATE TABLE CareCentre (
  CentreID                  smallint,
  Location                  varchar(150)  NOT NULL,
  NurseID                   smallint      NOT NULL,
  DateAssigned              DATE          NOT NULL,
  PRIMARY KEY  (CentreID),
  FOREIGN KEY (NurseID) REFERENCES Nurse(NurseID)
        ON DELETE RESTRICT
        ON UPDATE CASCADE
) ENGINE=InnoDB;
```

- have to insert into Nurse 1st, then into CareCentre
- query it by joining the Nurse and CareCentre tables

© University of Melbourne 2018

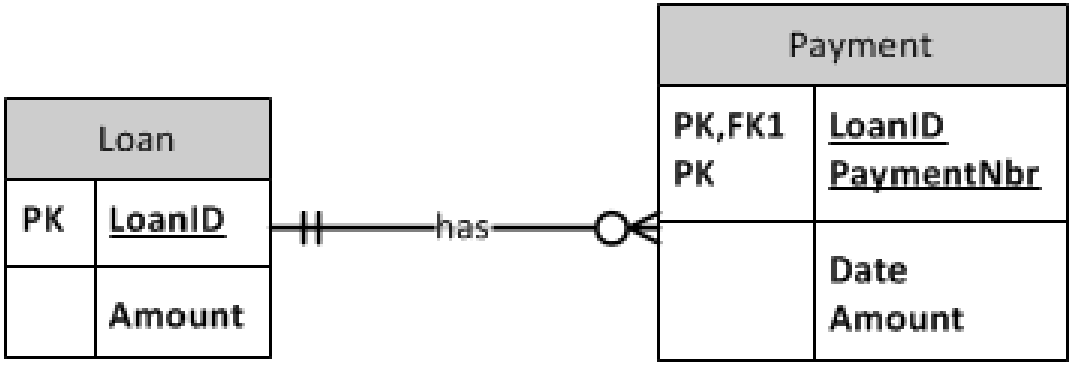
Summary of Binary Relationships

- One-to-Many
  - primary key on ONE side becomes foreign key on MANY side
- Many-to-Many
  - create an Associative Entity (a new table) with a compound primary key consisting of 2 FKs that refer to the other 2 tables
  - you then have two One-to-Many joins
- One-to-One
  - decide in which table to put the foreign key
  - foreign key on the optional side refers to primary key on the mandatory side

© University of Melbourne 2018

1-M special case – "Identifying Relationship"

- How to deal with an Identifying relationship
  - i.e. a relationship between weak child and strong parent tables
  - Foreign Key defines the relationship at the crows foot end.
  - and FK becomes part of the Primary Key

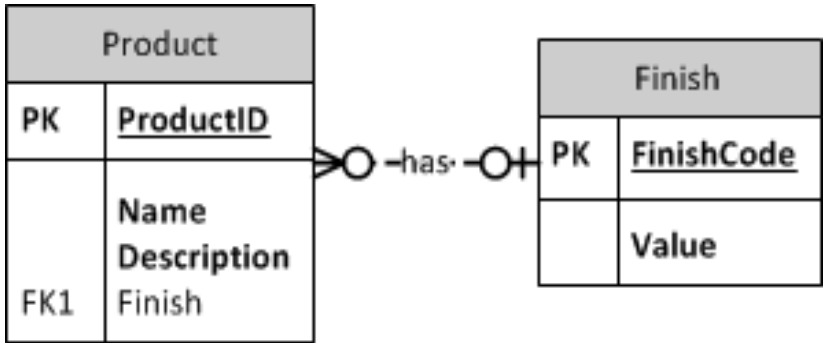


  - Logical Design
    - Loan = (LoanID, Amount)
    - Payment = (LoanID, PaymentID, Date, Amount)
  - Physical Design = normal one-to-many relationship

© University of Melbourne 2018

1-M special case – "Lookup table"

- Consider the following logical design



ProductID	Nam	…	Finish
1	Chai		A
2	Desk		C
3	Tabl		B
4	Book		A

Code	Value
A	Birch
B	Maple
C	Oak

- Physical design decision
  - Implement as 2 tables or one? trade-off = speed vs data integrity

ProductID	Name	…	Finish
1	Chair		Birch
2	Desk		Oak
3	Table		Maple
4	Bookcase		Birch

© University of Melbourne 2018

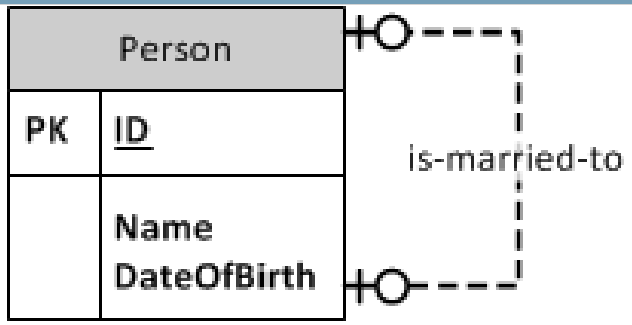
Unary Relationships

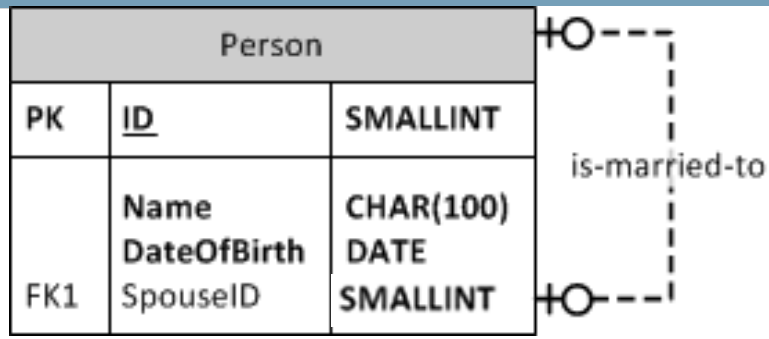
Unary Relationships

- Operate in the same way exactly as binary relationships
  - One-to-One
    - put a Foreign key in the entity
  - One-to-Many
    - put a Foreign key in the entity
  - Many-to-Many
    - create an extra table - Associative Entity
    - put two Foreign keys in the Associative Entity
      - the two FKs need different names
      - the FKs become the combined PK of the Associative Entity

© University of Melbourne 2018

Unary – One-to-One





Logical Design

Physical Design

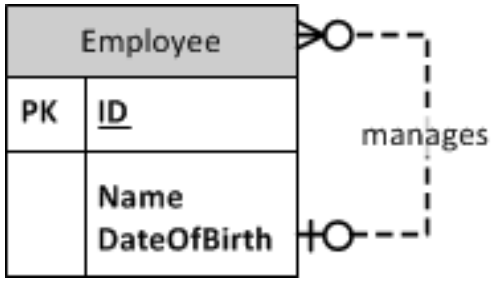
(ID, Name, DateOfBirth, SpouseID)

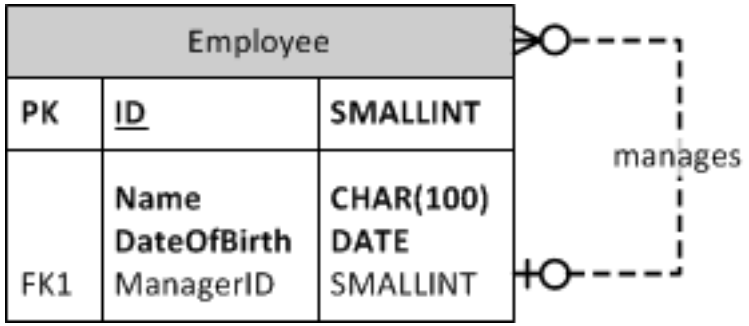
ID	Name	DOB	SpouseID
1	Ann	1969-06-12	3
2	Fred	1971-05-09	
3	Chon	1982-02-10	1
4	Nancy	1991-01-01	

```
CREATE TABLE Person (
   ID                  smallint,
   Name                varchar(150)  NOT NULL,
   DateOfBirth         DATE          NOT NULL,
   SpouseID            smallint      NOT NULL,
   PRIMARY KEY  (ID),
   FOREIGN KEY (SpouseID) REFERENCES Person(ID)
         ON DELETE RESTRICT
         ON UPDATE CASCADE
) ENGINE=InnoDB;
```


© University of Melbourne 2018

Unary – One-to-Many





Logical Design

(ID, Name, DateOfBirth, ManagerID)

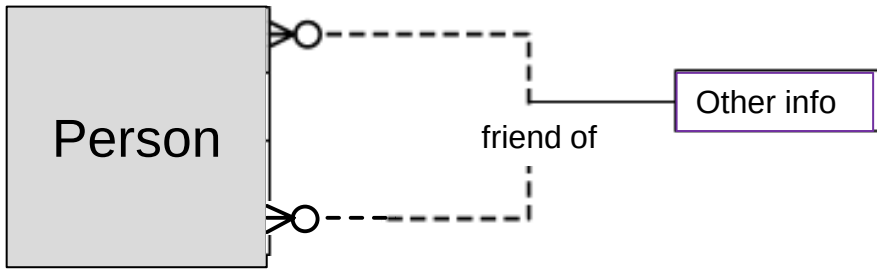
Physical Design

```
CREATE TABLE Employee (
    ID                  smallint,
    Name                varchar(150)  NOT NULL,
    DateOfBirth         DATE          NOT NULL,
    ManagerID           smallint      NOT NULL,
    PRIMARY KEY  (ID),
    FOREIGN KEY (ManagerID) REFERENCES Employee(ID)
          ON DELETE RESTRICT
          ON UPDATE CASCADE
) ENGINE=InnoDB;
```

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

© University of Melbourne 2018

Unary – Many-to-Many

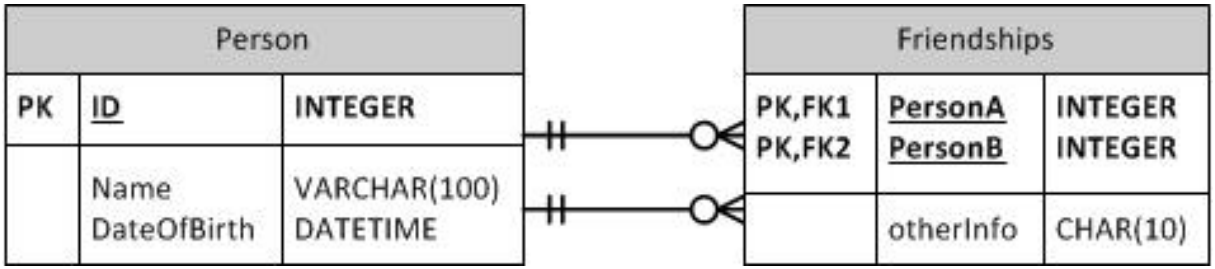


- Logical Design
  - Set up Associative Entity as for any M-M relationship
    - Person = (ID, Name, DateOfBirth)
    - Friendship = (PersonA, PersonB, otherInfo)

© University of Melbourne 2018

Unary – Many-to-Many

- Physical Design



- Implementation

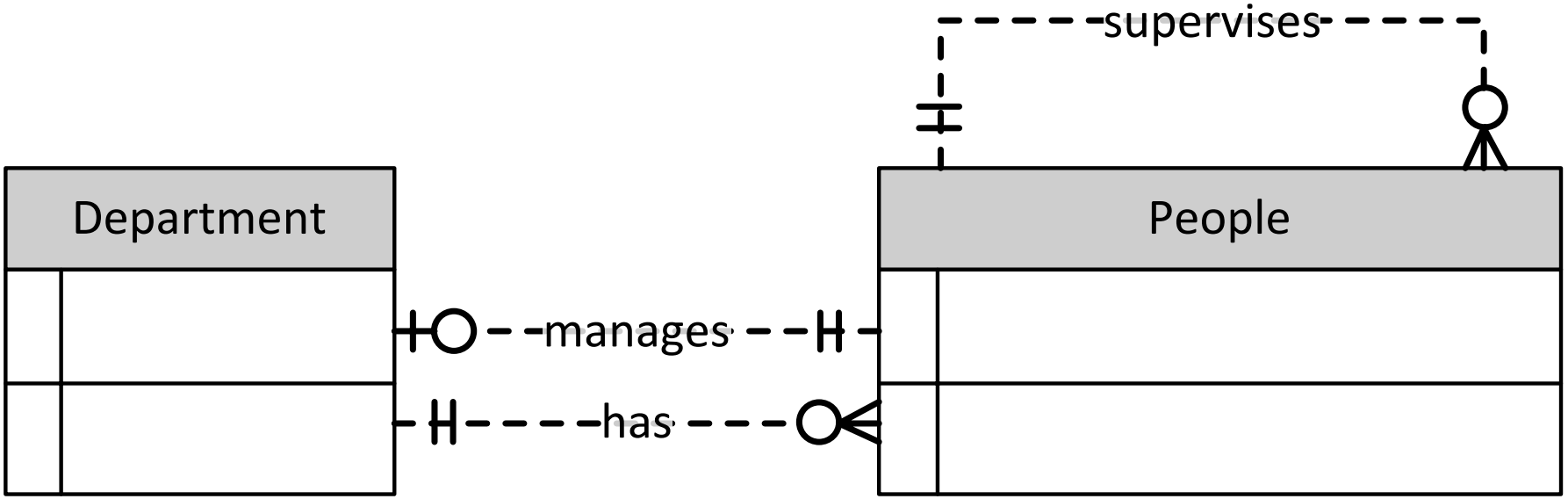
```
-- -----------------------------------------------------
-- Table `mydb`.`Person`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Person` (
  `ID` INT NOT NULL,
  `Name` VARCHAR(50) NULL,
  `DateOfBirth` DATE NULL,
  PRIMARY KEY (`ID`))
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `mydb`.`Friendship`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Friendship` (
  `PersonA` INT NOT NULL,
  `PersonB` INT NOT NULL,
  `otherInfo` CHAR(10) NULL,
  PRIMARY KEY (`PersonA`, `PersonB`),
  INDEX `fk_Friendship_Person1_idx` (`PersonB` ASC),
  CONSTRAINT `fk_Friendship_Person`
    FOREIGN KEY (`PersonA`)
    REFERENCES `mydb`.`Person` (`ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Friendship_Person1`
    FOREIGN KEY (`PersonB`)
    REFERENCES `mydb`.`Person` (`ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```


© University of Melbourne 2018

Multiple Relationships

- Entities can be related in several ways simultaneously



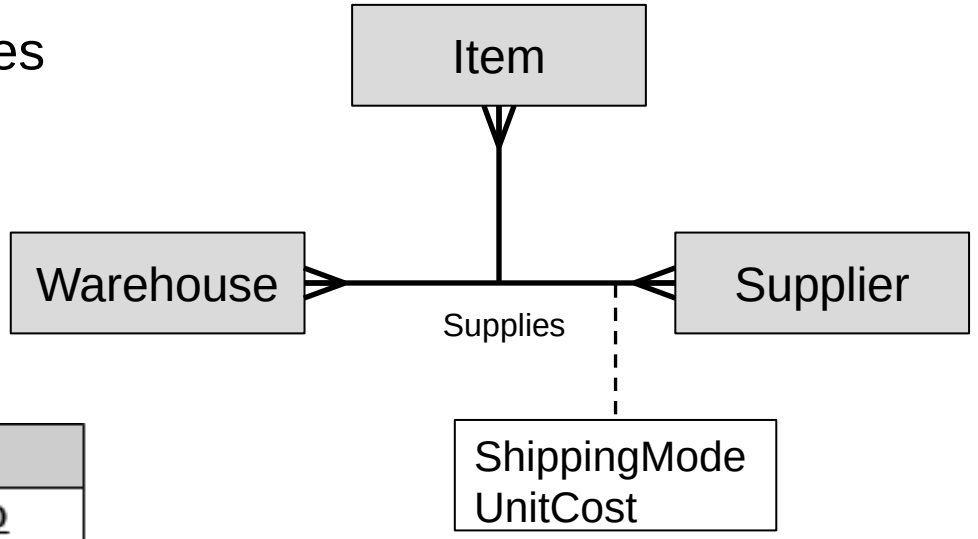
- Treat this the same was as any other One-to-Many, One-to-One relationship

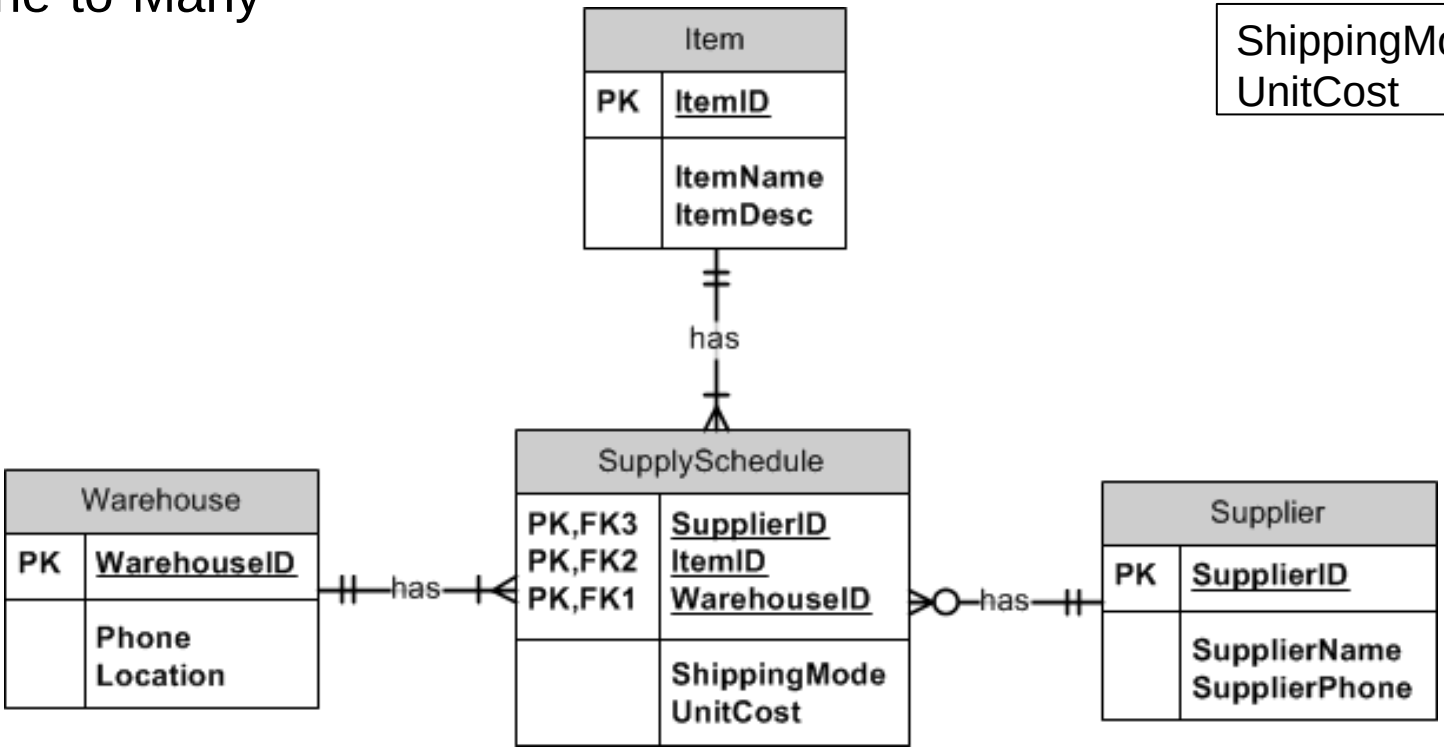
© University of Melbourne 2018

Ternary Relationships

Ternary relationships

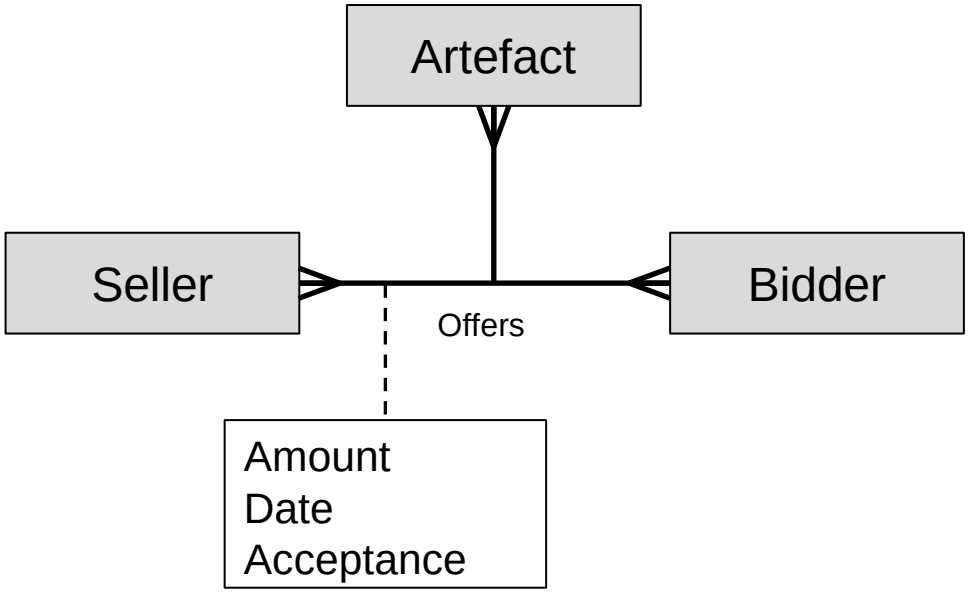
- Relationships between three entities
- Generate an Associative Entity
- Set up three One-to-Many relationships
- These are like any other One-to-Many





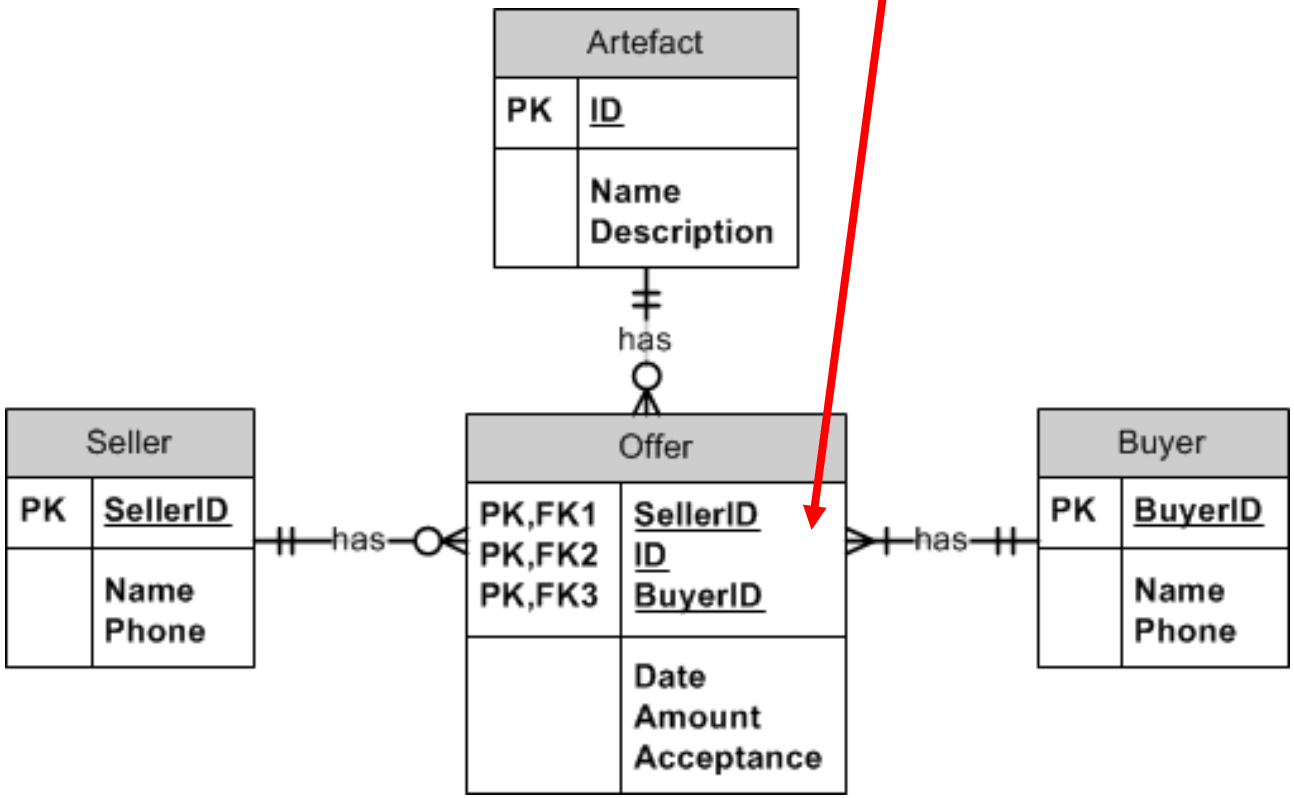
© University of Melbourne 2018

Ternary example – auction



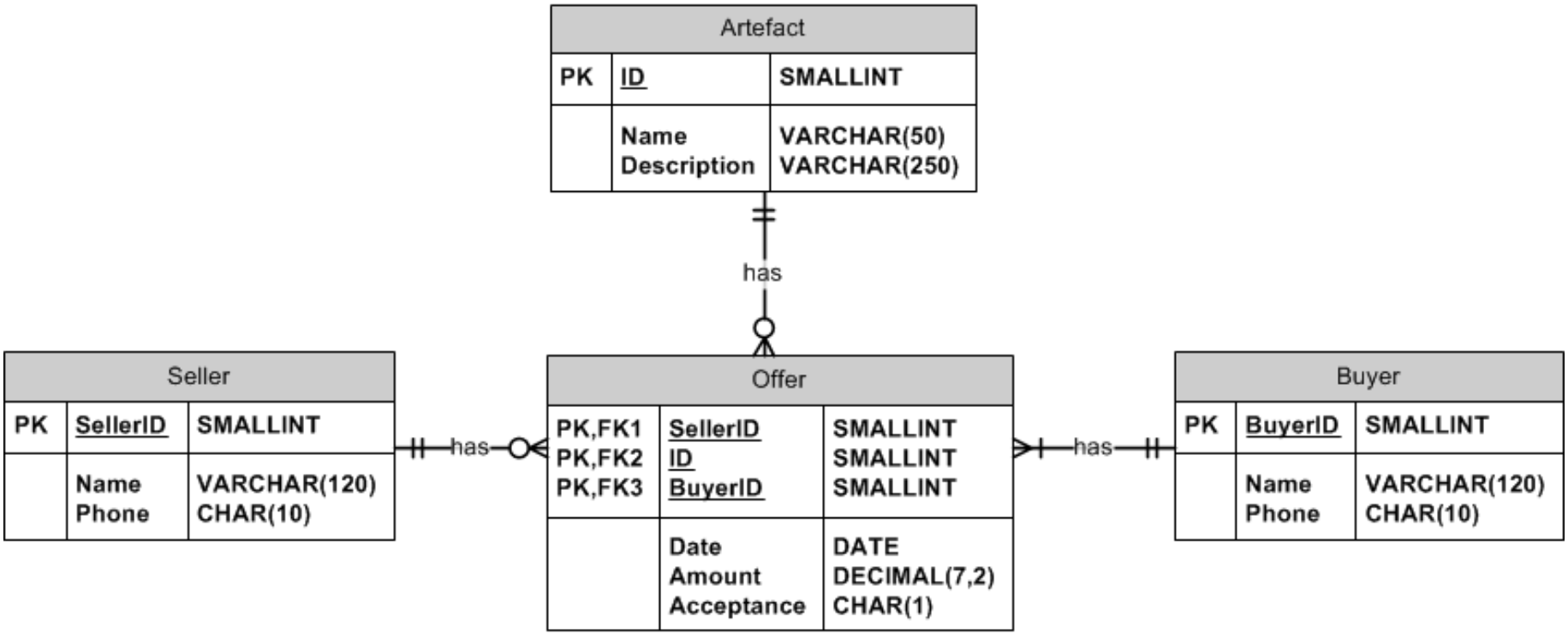
BUSINESS RULES:

- an Artefact can be sold several times by several different Sellers (over time)
- each Buyer can only make 1 offer to buy per Seller/Artefact combination



© University of Melbourne 2018

Auction Bids - Physical



© University of Melbourne 2018

Auction Bids – Table Creation

```
CREATE TABLE Seller (
  SellerID          smallint,
  Name              varchar(120) NOT NULL,
  Phone             char(10)     NOT NULL,
  PRIMARY KEY  (SellerID)
) ENGINE=InnoDB;
```

```
CREATE TABLE Buyer (
  BuyerID          smallint,
  Name              varchar(120) NOT NULL,
  Phone             char(10)     NOT NULL,
  PRIMARY KEY  (BuyerID)
) ENGINE=InnoDB;
```

```
CREATE TABLE Artefact (
  ID              smallint,
  Name            varchar(50)     NOT NULL,
  Description     varchar(250)    NOT NULL,
  PRIMARY KEY  (ID)
) ENGINE=InnoDB;
```


© University of Melbourne 2018

Auction Bids – Table Creation

```
CREATE TABLE Offer (
  SellerID          smallint      NOT NULL,
  ArtefactID        smallint      NOT NULL,
  BuyerID           smallint      NOT NULL,
  Date              DATE          NOT NULL,
  Amount            DECIMAL(12,2) NOT NULL,
  Acceptance        CHAR(1) NOT NULL DEFAULT "N",
  PRIMARY KEY  (SellerID, ArtefactID, BuyerID),
  FOREIGN KEY (ArtefactID) REFERENCES Artefact(ID)
        ON DELETE RESTRICT
        ON UPDATE CASCADE,
  FOREIGN KEY (SellerID) REFERENCES Seller(SellerID)
        ON DELETE RESTRICT
        ON UPDATE CASCADE,
  FOREIGN KEY (BuyerID) REFERENCES Buyer(BuyerID)
        ON DELETE RESTRICT
        ON UPDATE CASCADE
) ENGINE=InnoDB;
```


© University of Melbourne 2018

Auction Bids – Data Creation

```
INSERT INTO Seller VALUES (1, "Abby", "0233232232");
INSERT INTO Seller VALUES (2, "Ben", "0311111111");

INSERT INTO Buyer VALUES (1, "Maggie", "0333333333");
INSERT INTO Buyer VALUES (2, "Nicole", "0444444444");

INSERT INTO Artefact VALUES (1, "Vase", "Old Vase");
INSERT INTO Artefact VALUES (2, "Knife", "Old Knife");

INSERT INTO Offer VALUES (1, 1, 1, "2012-06-20", 81223.23, DEFAULT);
INSERT INTO Offer VALUES (1, 1, 2, "2012-06-20", 82223.23, DEFAULT);
INSERT INTO Offer VALUES (2, 2, 1, "2012-06-20", 19.95, DEFAULT);
INSERT INTO Offer VALUES (2, 2, 2, "2012-06-20", 23.00, DEFAULT);
```

- list all Offers. Show Artefact, Seller, Buyer and Offer details
- this is a FOUR table join

```
SELECT * FROM Artefact
       INNER JOIN Offer ON Artefact.ID = Offer.ArtefactID
       INNER JOIN Seller ON Seller.SellerID = Offer.SellerID
       INNER JOIN Buyer ON Buyer.BuyerID = Offer.BuyerID;
```


© University of Melbourne 2018

Ternary Query Output

ID	Name	Description	SellerID	ArtefactID	BuyerID	Date	Amount	Accep	SellerID	Name	Phone	BuyerID	Name	Phone
1	Vase	Old Vase	1	1	1	2012-06-20	81223.23	N	1	Abby	0233232232	1	Maggie	0333333333
1	Vase	Old Vase	1	1	2	2012-06-20	82223.23	N	1	Abby	0233232232	2	Nicole	0444444444
2	Knife	Old Knife	2	2	1	2012-06-20	19.95	N	2	Ben	0311111111	1	Maggie	0333333333
2	Knife	Old Knife	2	2	2	2012-06-20	23.00	N	2	Ben	0311111111	2	Nicole	0444444444

- Note the value of Accepted
  - "N" – the default value from our create statement
- Note that some columns have ambiguous names
  - SellerID
  - BuyerID
  - Name
  - Phone

© University of Melbourne 2018

Better output by using aliases

```
SELECT A.ID, A.Name AS Artefact, A.Description AS ArtDesc, Date AS OfferDate,
       Amount AS OfferAmount, Acceptance AS OfferAccepted, S.SellerID,
       S.Name AS Seller, S.Phone AS SellerPhone, B.BuyerID, B.Name AS Buyer,
       B.Phone AS BuyerPhone
       FROM Artefact A
            INNER JOIN Offer O ON A.ID = O.ArtefactID
            INNER JOIN Seller S ON S.SellerID = O.SellerID
            INNER JOIN Buyer B ON B.BuyerID = O.BuyerID;
```

ID	Artefact	ArtDesc	OfferDate	OfferAmount	OfferAccepted	SellerID	Seller	SellerPhone	BuyerID	Buyer	BuyerPhone
1	Vase	Old Vase	2012-06-20	81223.23	N	1	Abby	0233232232	1	Maggie	0333333333
1	Vase	Old Vase	2012-06-20	82223.23	N	1	Abby	0233232232	2	Nicole	0444444444
2	Knife	Old Knife	2012-06-20	19.95	N	2	Ben	0311111111	1	Maggie	0333333333
2	Knife	Old Knife	2012-06-20	23.00	N	2	Ben	0311111111	2	Nicole	0444444444

- aliases for table names: “A” “O” “S” “B”
- aliases for column names: Artefact, ArtDesc etc

© University of Melbourne 2018