Dr Greg Wadley

INFO 90002
Database Systems & Information Modelling

Week 04

Data Modelling and SQL (2)

Today's Session…

- New kinds of relationships in data models
  - Many to Many
    - Associative entity
  - One to One
  - Recursive / Unary relationships
    - One-to-one, One-to-many, Many-to-Many
  - Ternary relationships
    - 3 tables are involved
  - Multiple One to Many Relationships
    - Special cases of one-to-many
- New SQL
  - Referential and Data Field Integrity
  - Rules for FK's – CASCADE, RESTRICT etc.
  - Nested queries

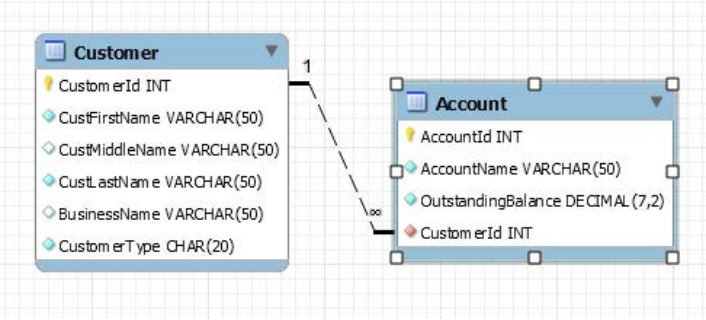
Recap: joining 2 tables

- Data about an entity is spread across 2 tables
- Inner join = Join rows where FK value = PK value

```
select * from Customer inner join Account
on Customer.CustomerId = Account.CustomerId;
```

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

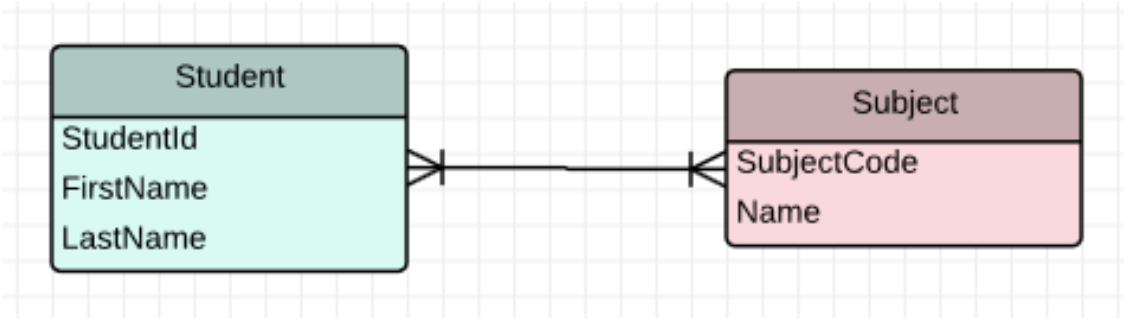
CustID	CustFirstName	CustMiddleName	CustLastName	BusinessName	CustType
1	Peter		Smith		Personal
2	James		Jones	JJ Enterprises	Company



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

Many to Many relationships

- 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?

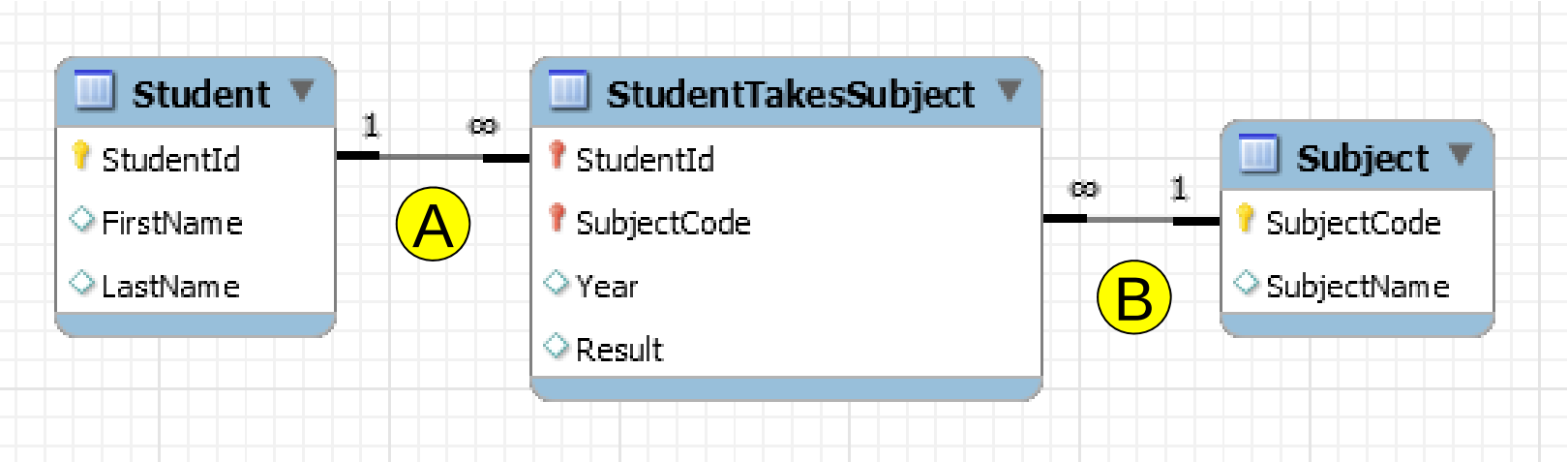


StudentId	FirstName	LastName
11111	John	Lennon
22222	Paul	McCartney
33333	George	Harrison

SubjCode	Name
INFO90002	Database
ISYS90026	Fundamentals
ISYS90081	Organisational

Problems modelling Many-Many…

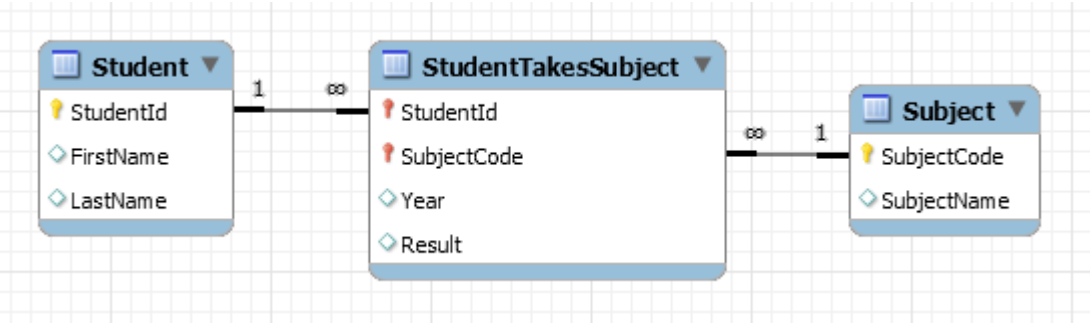
- Relational database doesn't directly support M-M…
  - So we create an Associative Entity between the other 2 entities (when converting Conceptual to Logical model)
  - Each of these 2 relationships is like any 1-M relationship



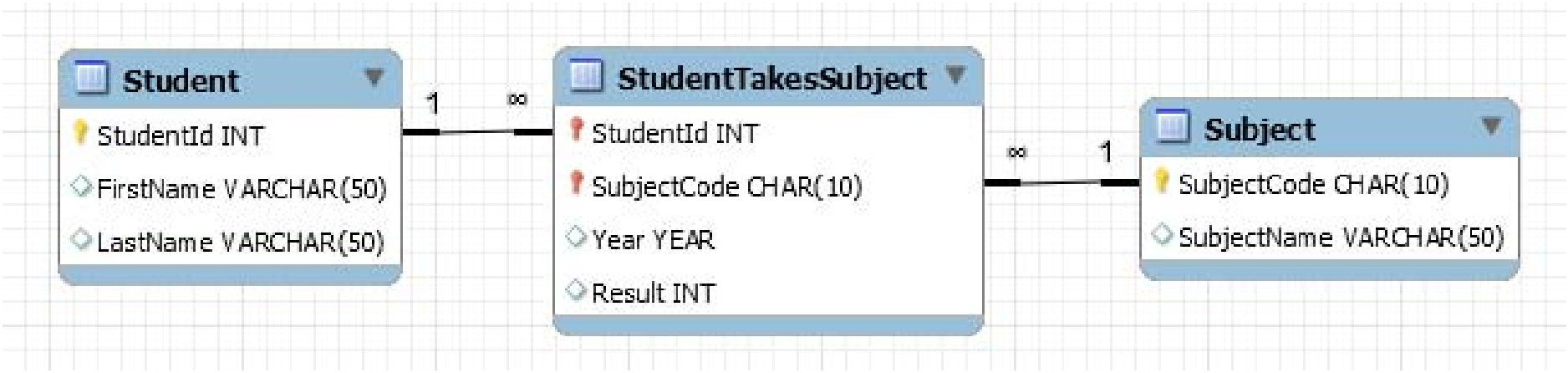
  - We can add attributes to the associative entity to record when the student took the subject and the result they got.
  - Associate Entities are called Join Tables and many other names, see https://en.wikipedia.org/wiki/Junction_table

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 than the FKs
  - may participate in other relationships



Many-Many Physical Model



- Work out data types
- Example decisions:
  - are StudentIds number or strings?
  - How long are people's names?
  - Are results integers or floating point?

Many-Many CREATE statements

- Order of creation is important!
  - So is order of deletion…
- Create tables *without* foreign keys first
  - Delete 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;
```

Many-Many CREATE statements

- Order of creation is important!
  - So is order of deletion…
- Create tables *with* foreign keys last
  - Delete 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;
```

Data Integrity Constraints

- Domain Integrity
  - Valid values and domain
    - Selection of data type is the initial constraint on the data
  - 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

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)
```

Adding data to a M-M relationship

- Insert into the joint 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);
```

How to read complete student results

- Three table join

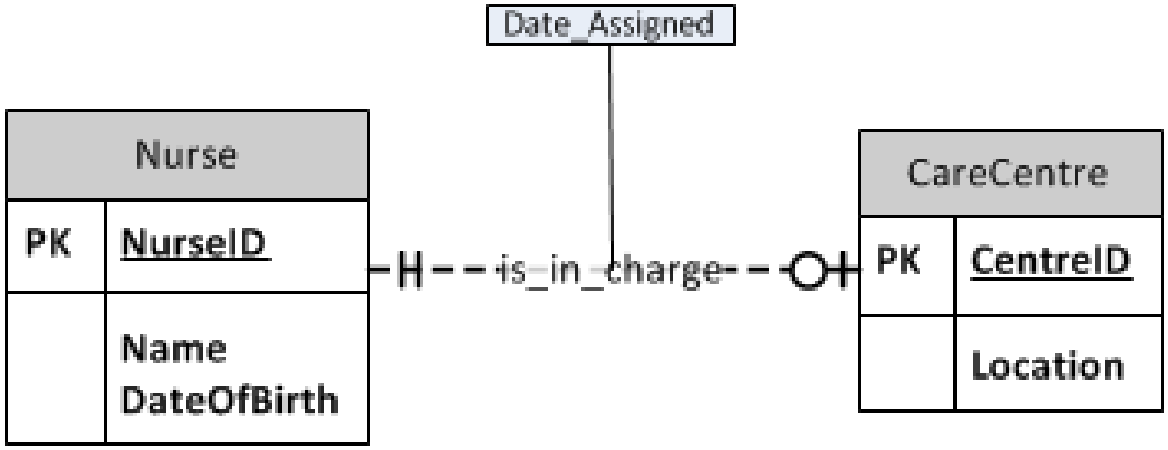
```
1 • select * from
2   Student natural join StudentTakesSubject
3   natural join Subject;
```

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...

Binary One-One Relationship

- Given this example… How do we implement it…

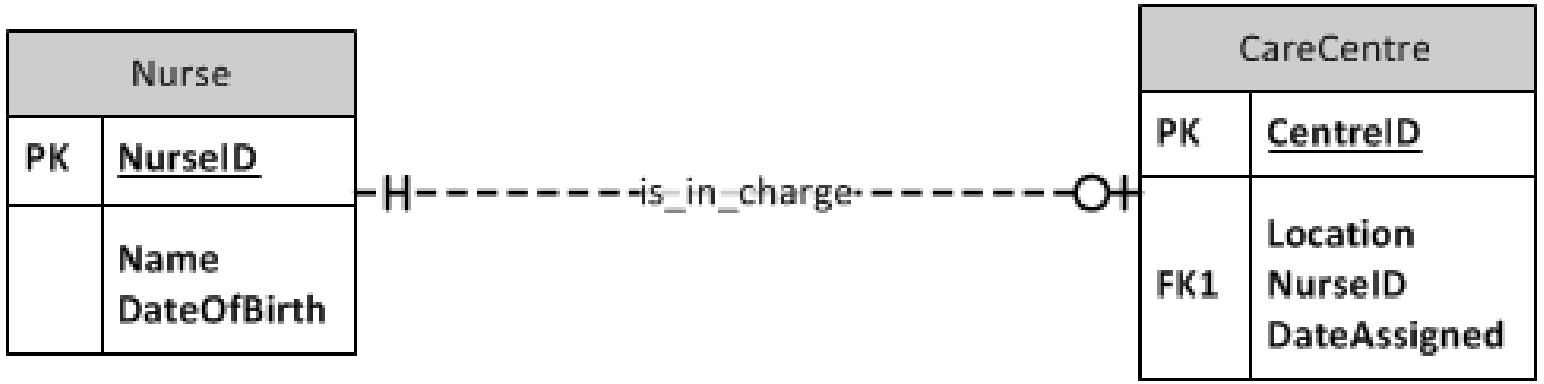


- Note: Date_assigned is an attribute of the relationship

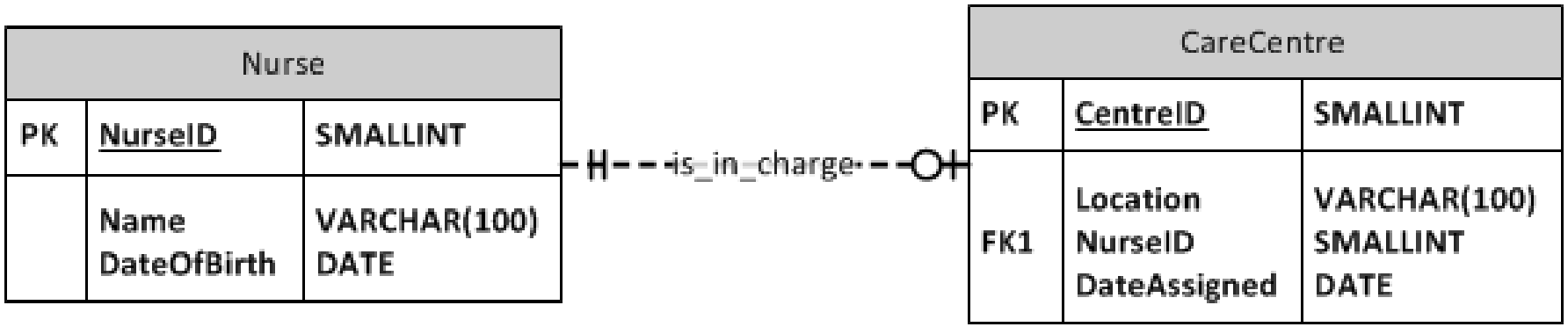
- Need to decide whether to put the foreign key inside Nurse or CareCentre (in which case you would have the Date_Assigned in the same location)
  - Where would the least NULL values be?
  - The rule is the OPTIONAL side of the relationship gets the foreign key

Binary One-One Relationship – Logical and Physical

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



- Physical



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;
```

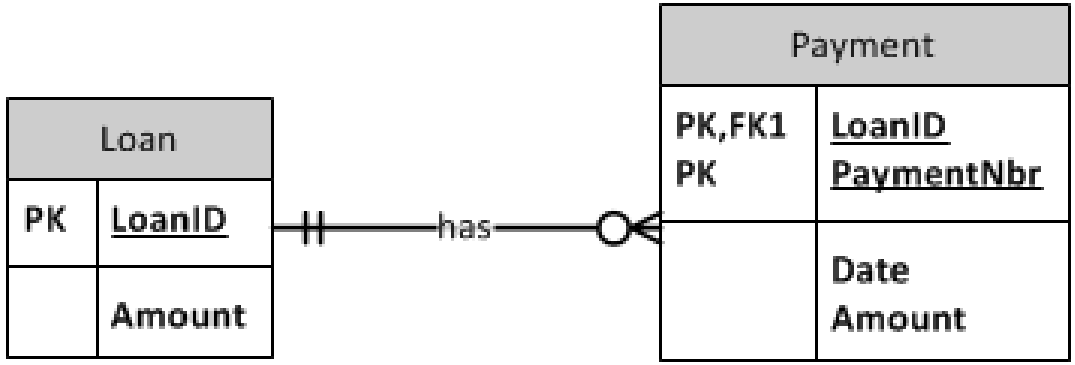
- Have to insert into Nurse 1st, then create CareCentre Records
- Query it by joining the Nurse and CareCentre tables

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

1-M special case – "Identifying Relationship"

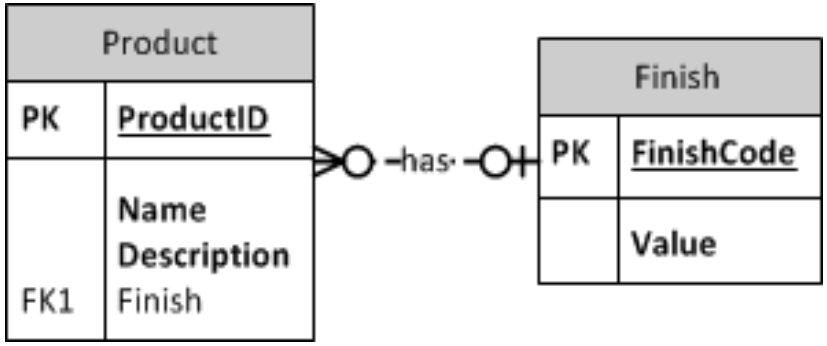
- How to deal with an Identifying relationship
  - i.e. a relationship between weak child and strong parent tables
  - Foreign Key defines the relationship at the crows foot end.
  - the difference = FK becomes part of the Primary Key



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

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

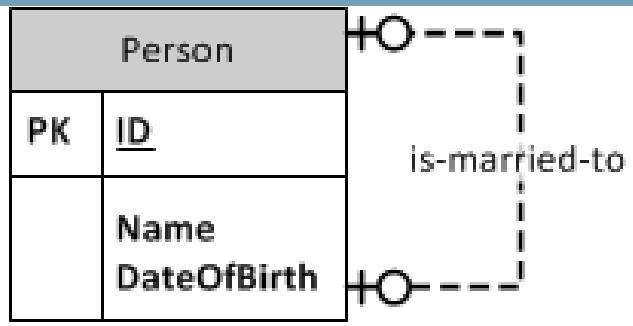
- Physical design decision
  - Implement as 2 tables or one? trade-off = speed vs data integrity

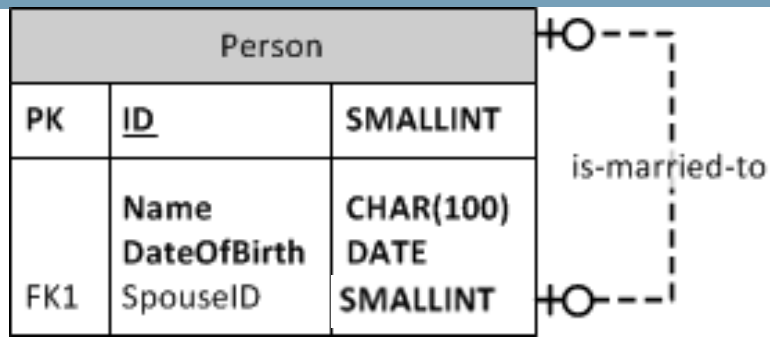
ProductID	Name	…	Finish
1	Chair		Birch
2	Desk		Oak
3	Table		Maple
4	Bookcase		Birch

Unary Relationships

- Operate in the same way exactly 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
    - Create an extra table - Associative Entity
    - Put two Foreign keys in the Associative Entity
      - Need different names for the two FKs
      - FKs become the combined PK of the Associative Entity

Unary – One-to-One





Logical Design

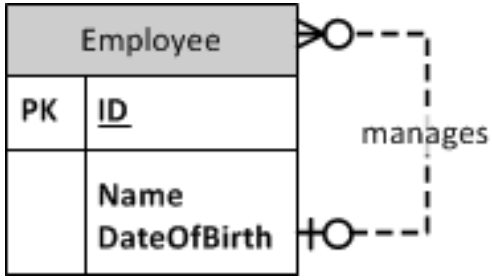
Physical Design

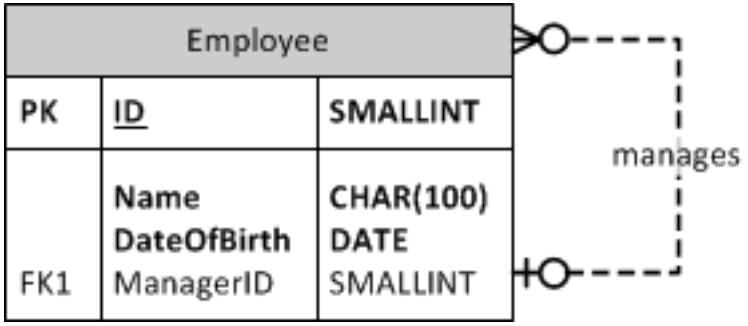
- Person = (ID, Name, DateOfBirth, SpouseID)

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;
```

Unary – One-to-Many





Logical Design

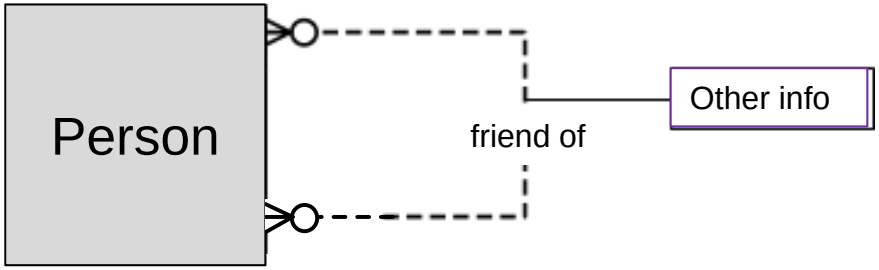
- Employee = (ID, Name, DateOfBirth, ManagerID)

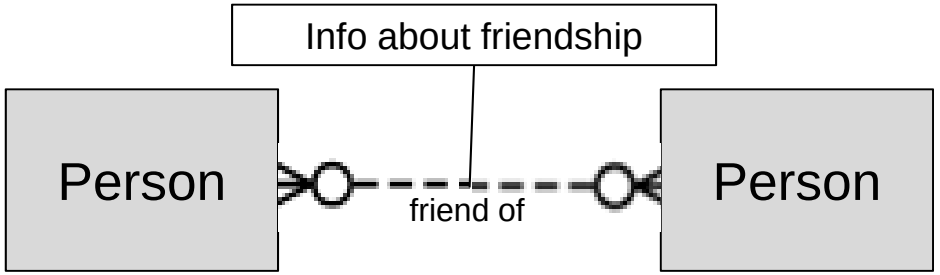
ID	Name	DOB	MngrID
1	Ann	1969-06-12	
2	Fred	1971-05-09	1
3	Chon	1982-02-10	1
4	Nancy	1991-01-01	1

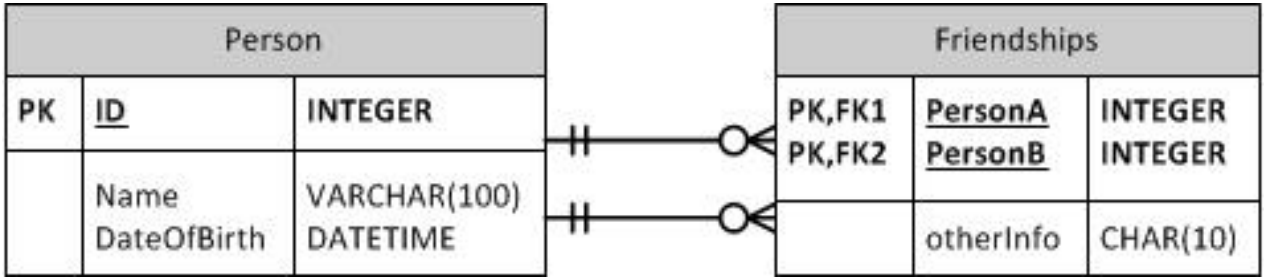
Physical Design

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

Unary – Many-to-Many



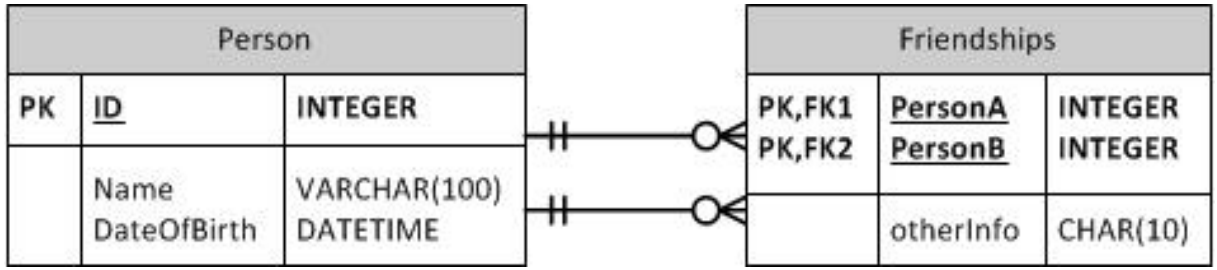




- Logical Design
  - Set up Associative Entity as for any M-M relationship
    - Person = (ID, Name, DateOfBirth)
    - Friendship = (PersonA, PersonB, otherInfo)

Unary – Many-to-Many

- Physical Design



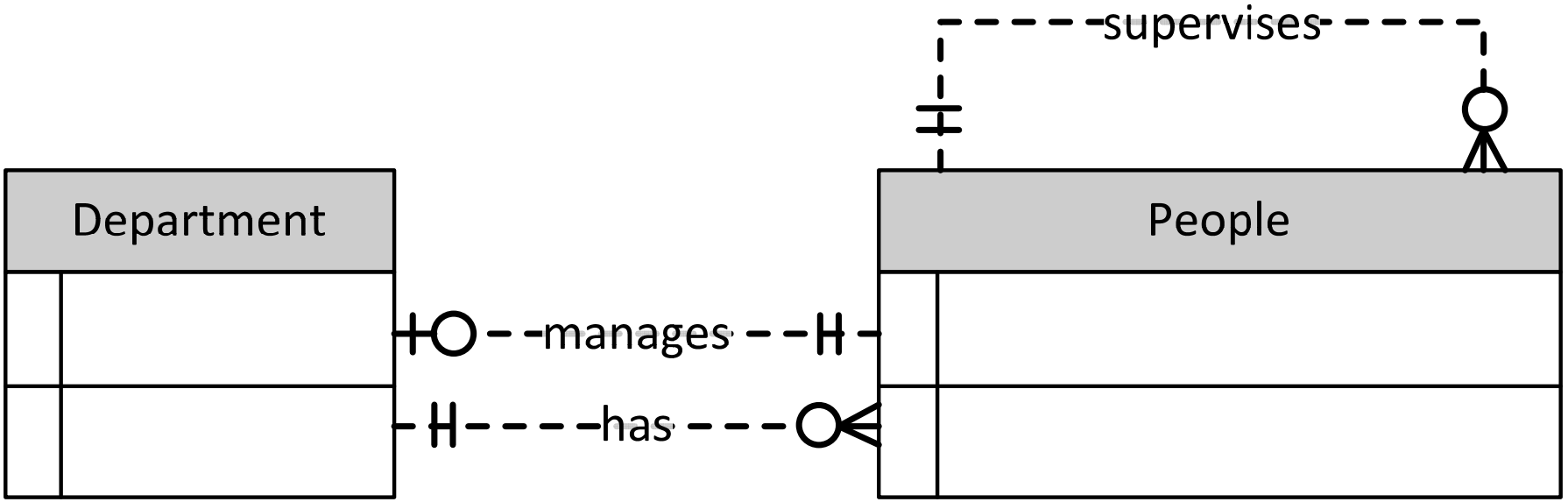
- Implementation

```
-- -----------------------------------------------------
-- 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;
```

Multiple Relationships

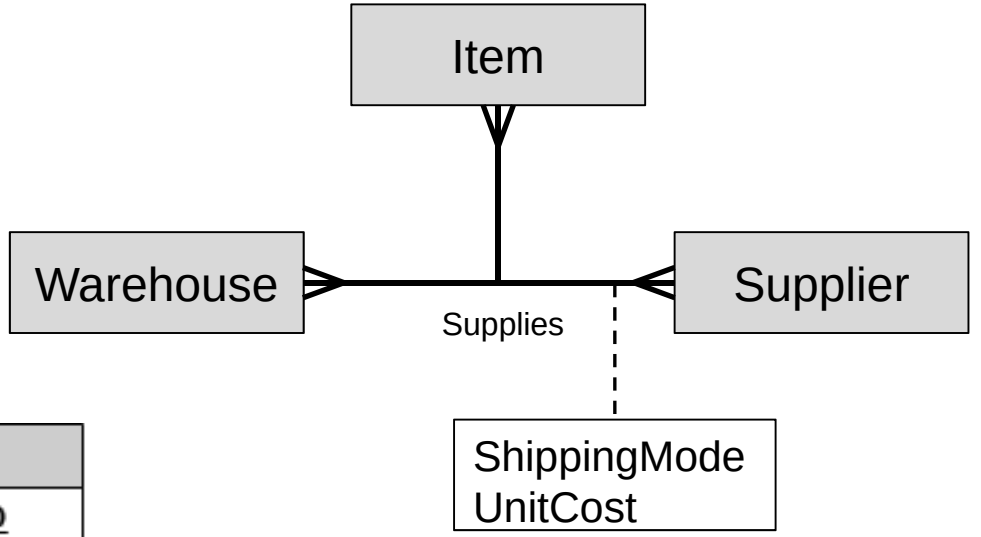
- Entities can be related in several ways simultaneously

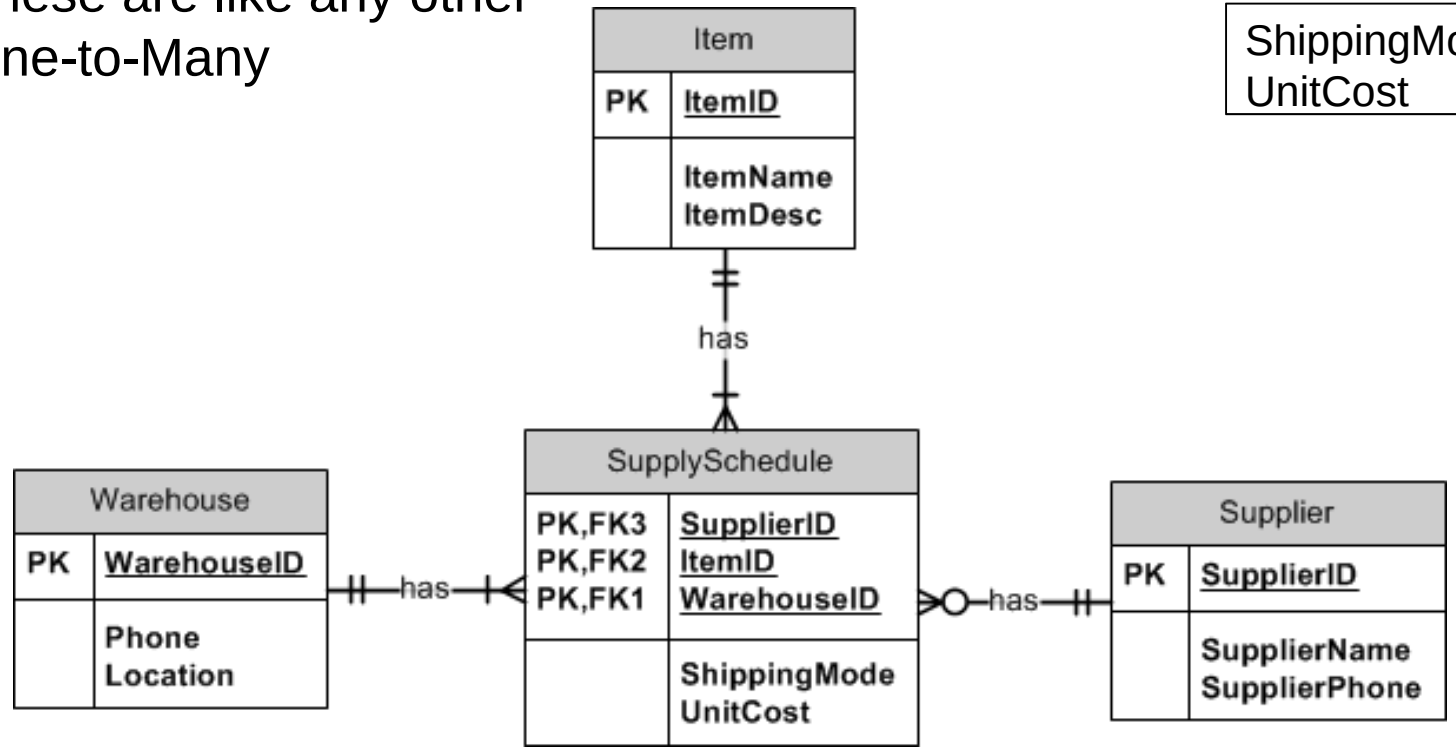


- Treat this the same was as any other One-to-Many, One-to-One relationship

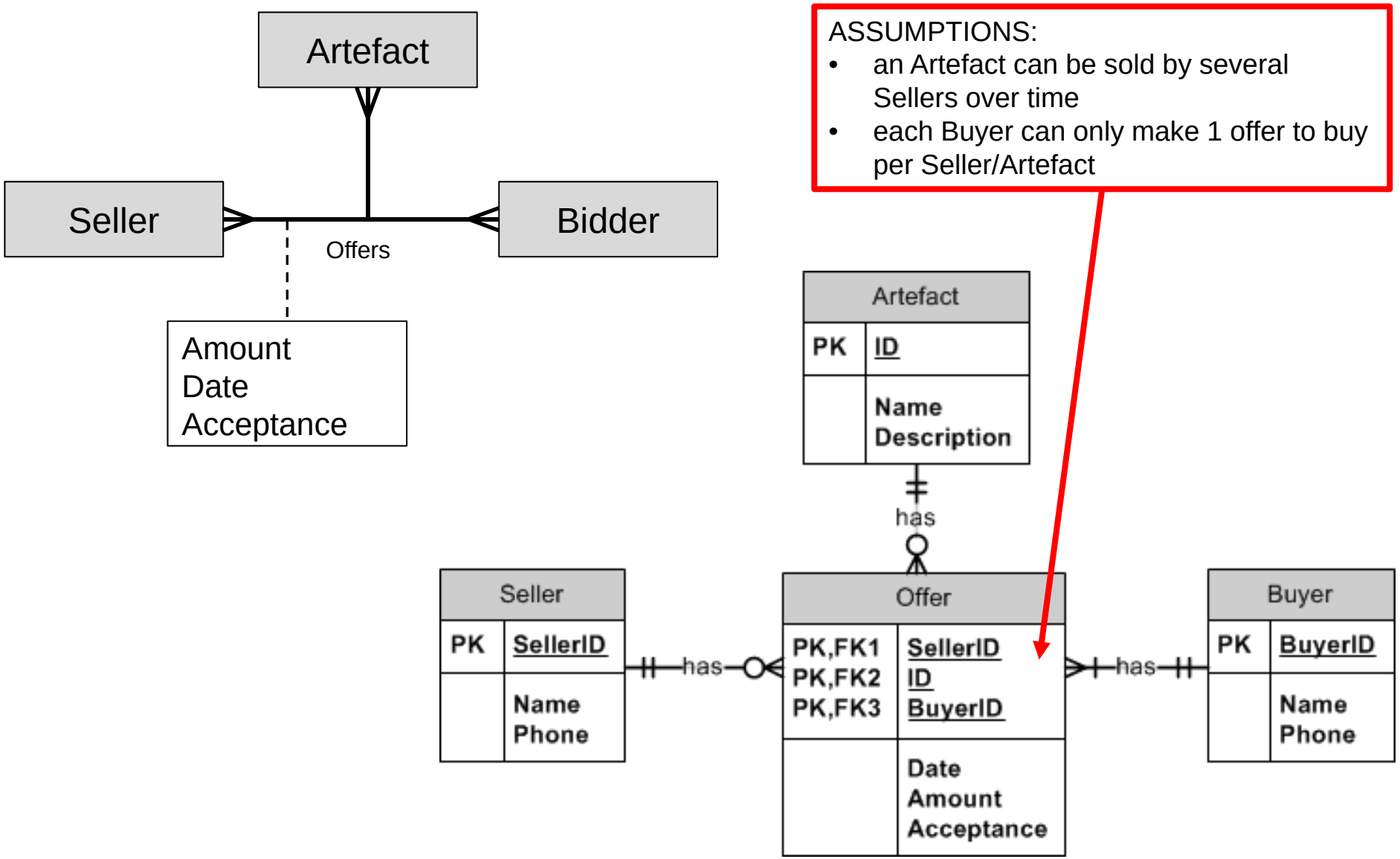
Ternary relationships

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

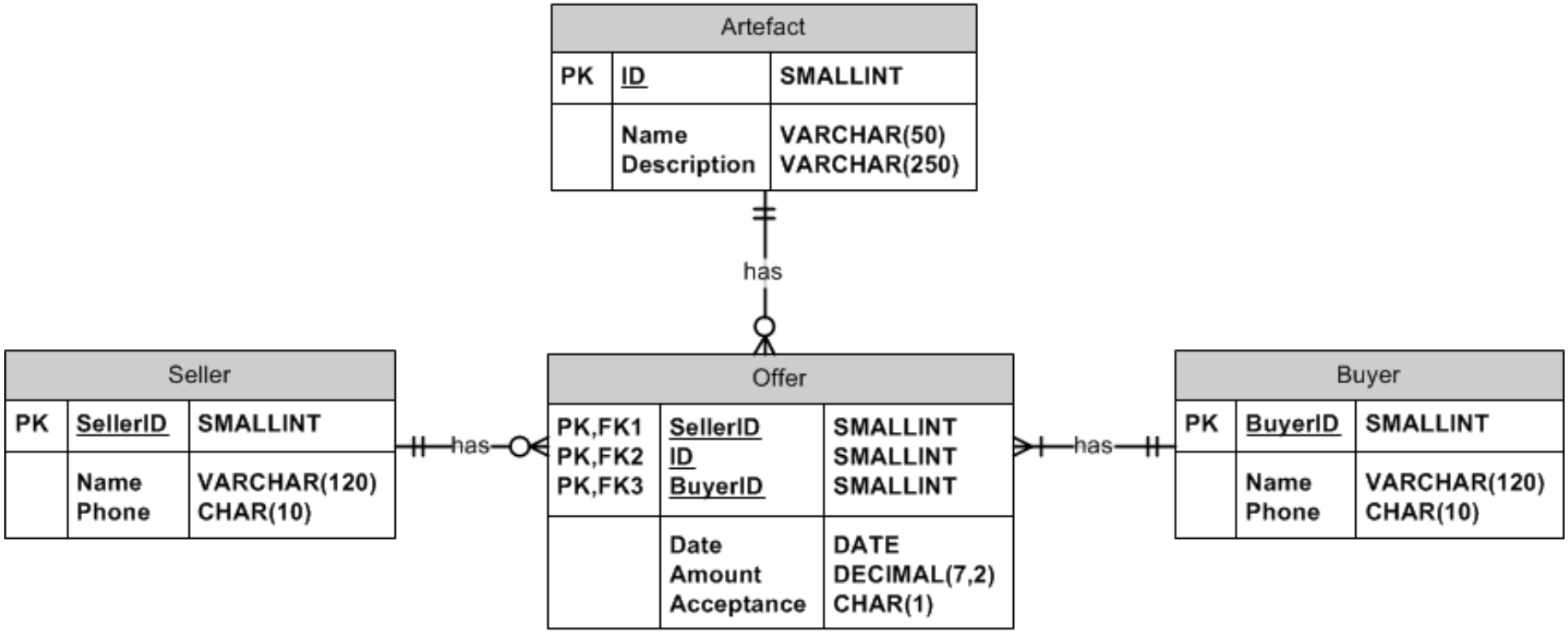




Ternary example – eBay auction



Auction Bids - Physical



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;
```

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;
```

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;
```

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

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

SQL – Subqueries / nested queries

- Select allows you to nest *sub-queries* inside the main or “outer” query
- A nested query is simply another Select query you write to produce a table of data
  - Remember that all select queries return a “table”
- A common use of sub-queries is to perform tests
  - Set membership, set comparisons
- Often there is an equivalent Join query
- Put the subquery inside round brackets

```
SELECT DISTINCT ItemID FROM Sale
WHERE DepartmentID IN
(SE                                    tment
                                       – 2);
```

(3, 7)

Comparison Operators for SubQueries

- IN / NOT IN
  - is the value a member of the set returned by the Subquery?
- ALL
  - true if all values returned meet the condition
- WHERE [NOT] EXISTS
  - true if the subquery yields any [/ no] results

Set Comparison examples

- auction example: Buyer, Seller, Artefact, Offer tables

ID	Name	Description
1	Vase	Old Vase
2	Knife	Old Knife
3	Pot	Old Pot

SellerID	Name	Phone
1	Abby	0233232232
2	Ben	0311111111
3	Carl	0333333333

BuyerID	Name	Phone
1	Maggie	0333333333
2	Nicole	0444444444
3	Oleg	0555555555

SellerID	ArtefactID	BuyerID	Date	Amount	Acceptance
1	1	1	2012-06-20	81223.23	N
1	1	2	2012-06-20	82223.23	N
2	2	1	2012-06-20	19.95	N
2	2	2	2012-06-20	23.00	N

Set queries

- Which Artefacts don't have offers made on them

```
SELECT * FROM Artefact
    WHERE ID NOT IN
        (SELECT ArtefactID FROM Offer);
```

ID	Name	Description
3	Pot	Old Pot

- Which Buyers haven't made a bid for Artefact 3

```
SELECT * FROM Buyer
    WHERE BuyerID NOT IN
        (SELECT BuyerID FROM Offer
            WHERE ArtefactID = 3);
```

BuyerID	Name	Phone
1	Maggie	0333333333
2	Nicole	0444444444
3	Oleg	0555555555

- Which Buyers haven't made a bid for the "Pot" Artefact

```
SELECT * FROM Buyer
    WHERE BuyerID NOT IN
        (SELECT BuyerID FROM Offer
            WHERE ArtefactID IN
                (SELECT ID FROM Artefact
                    WHERE Name = "Pot"));
```

BuyerID	Name	Phone
1	Maggie	0333333333
2	Nicole	0444444444
3	Oleg	0555555555

Multiple subquery

- Which Buyers have made a bid for the “Knife” Artefact

```
SELECT * FROM Buyer
    WHERE BuyerID IN
        (SELECT BuyerID FROM Offer
            WHERE ArtefactID IN
                (SELECT ID FROM Artefact
                    WHERE Name = "Knife"));
```

BuyerID	Name	Phone
1	Maggie	0333333333
2	Nicole	0444444444

There is often an equivalent Join that will achieve the same result. The above is equivalent to:

```
select Buyer.*
from Buyer natural join Offer natural join Artefact
where Artefact.name = 'Knife'
```

Aggregate functions

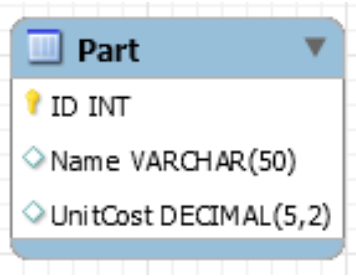
These functions operate on the multi-set of values in a column of a table and return a single value

- AVG()
  - Average value
- MIN()
  - Minimum value
- MAX()
  - Maximum value
- COUNT()
  - Number of values
- SUM()
  - Sum of values

- Plus others
  - http://dev.mysql.com/doc/refman/5.7/en/group-by-functions.html
- All of these except for COUNT() ignore null values and return null if all values are null. COUNT() counts the rows not the values and thus even if the value is NULL it is still counted.

Finding above average

- one table: “Part”



- Which parts have a UnitCost that is higher than the average?

```
SELECT * FROM Part PA
    WHERE PA.UnitCost > ALL
        (SELECT AVG(UnitCost) FROM Part);
```

ID	Name	UnitCost
3	Table	2343.00
4	Chair	1230.00
7	Copier	2343.00

Finding a maximum

- find the part with the highest cost

ID	Name	UnitCost
1	Nut	0.43
2	Washer	0.23
3	Table	2343.00
4	Chair	1230.00
5	Lamp	132.10
6	Magazine	19.32

```
SELECT * FROM Part PA
    WHERE PA.UnitCost > ALL
        (SELECT UnitCost FROM Part PB
            WHERE PA.ID != PB.ID);
```

ID	Name	UnitCost
3	Table	2343.00

```
SELECT * FROM Part
    WHERE UnitCost IN
        (SELECT MAX(UnitCost) FROM Part);
```

ID	Name	UnitCost
3	Table	2343.00

Finding a maximum

- find the max: what if two products have the same price?

```
SELECT * FROM Part PA
    WHERE PA.UnitCost > ALL
        (SELECT UnitCost FROM Part PB
            WHERE PA.ID != PB.ID);
```

	ID	Name	UnitCost
*	NULL	NULL	NULL

(now doesn't work!)

ID	Name	UnitCost
1	Nut	0.43
2	Washer	0.23
3	Table	2343.00
4	Chair	1230.00
5	Lamp	132.10
6	Magazine	19.32
7	Copier	2343.00

```
SELECT * FROM Part
    WHERE UnitCost IN
        (SELECT MAX(UnitCost) FROM Part);
```

	ID	Name	UnitCost
▶	3	Table	2343.00
	7	Copier	2343.00
*	NULL	NULL	NULL

Practice – Draw a Logical Model

- *A Bus Company owns a number of busses. Each bus is allocated to a particular route, although some routes may have several busses. Each route passes through a number of suburbs. One or more drivers are allocated to each stage of a route, which corresponds to a journey through some or all of the suburbs on a route. Some of the suburbs have a garage where busses are kept and each of the busses are identified by the registration number and can carry different numbers of passengers, since the vehicles vary in size and can be single or double-decked. Each route is identified by a route number and information is available on the average number of passengers carried per day for each route. Drivers have an employee number, name, address, and sometimes a telephone number.*
- Answers released next week…