Introduction to Database & Data Modeling

Week 12

- Implementing a Relational Schema in MySQL
  - Determining if NOT NULL is needed for a foreign key attribute
  - FOREIGN KEY constraint
  - Determining order of CREATE TABLE and INSERT statements
- Lookup Tables
- Additional Data Types
  - TINYINT
  - BOOLEAN
  - ENUM
- Data Type Attribute: AUTO_INCREMENT
- INSERT with values from another table
- Foreign key constraint con't
  - ON UPDATE
  - ON DELETE
- ALTER TABLE con't
- Transactions
  - Atomicity
  - Consistency
  - Isolation
  - Durability
- Transaction Control Language
  - Commit
  - Rollback
- Transactions in MySQL

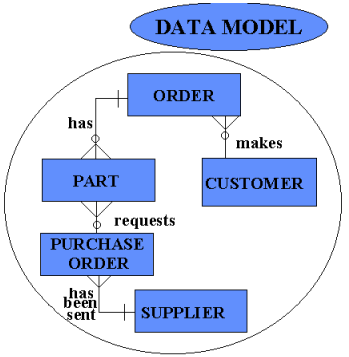
Expanded Database Development Lifecycle

- Summary of Steps
  1. **Create a Data Model from gathered requirements.**
     - **Entity-Relationship Model**
  2. Transpose Data Model into Relation(s)
  3. Normalize Relations
     A. Determine functional dependencies
     B. Determine candidate key(s)
     C. Confirm or modify "proposed" primary key
     D. Complete Normalization Process from First Normal Form (1NF) through Boyce-Codd Normal Form (BCNF) (per Normalization Process Reference sheet)
  4. Create the Relational Schema within Database Management System. (Metadata)
     - Create the database and the table(s)
  5. Define forms, queries, reports, menus, (Application Metadata), if supported within DBMS or external application programs.
  6. Populate database with User Data
  7. Maintenance

Review:

UoD

HIGH-LEVEL or *conceptual* level
(e.g., E-R diagram)



DBMS - INDEPENDENT

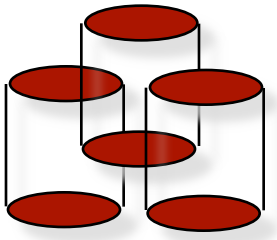
DBMS - SPECIFIC

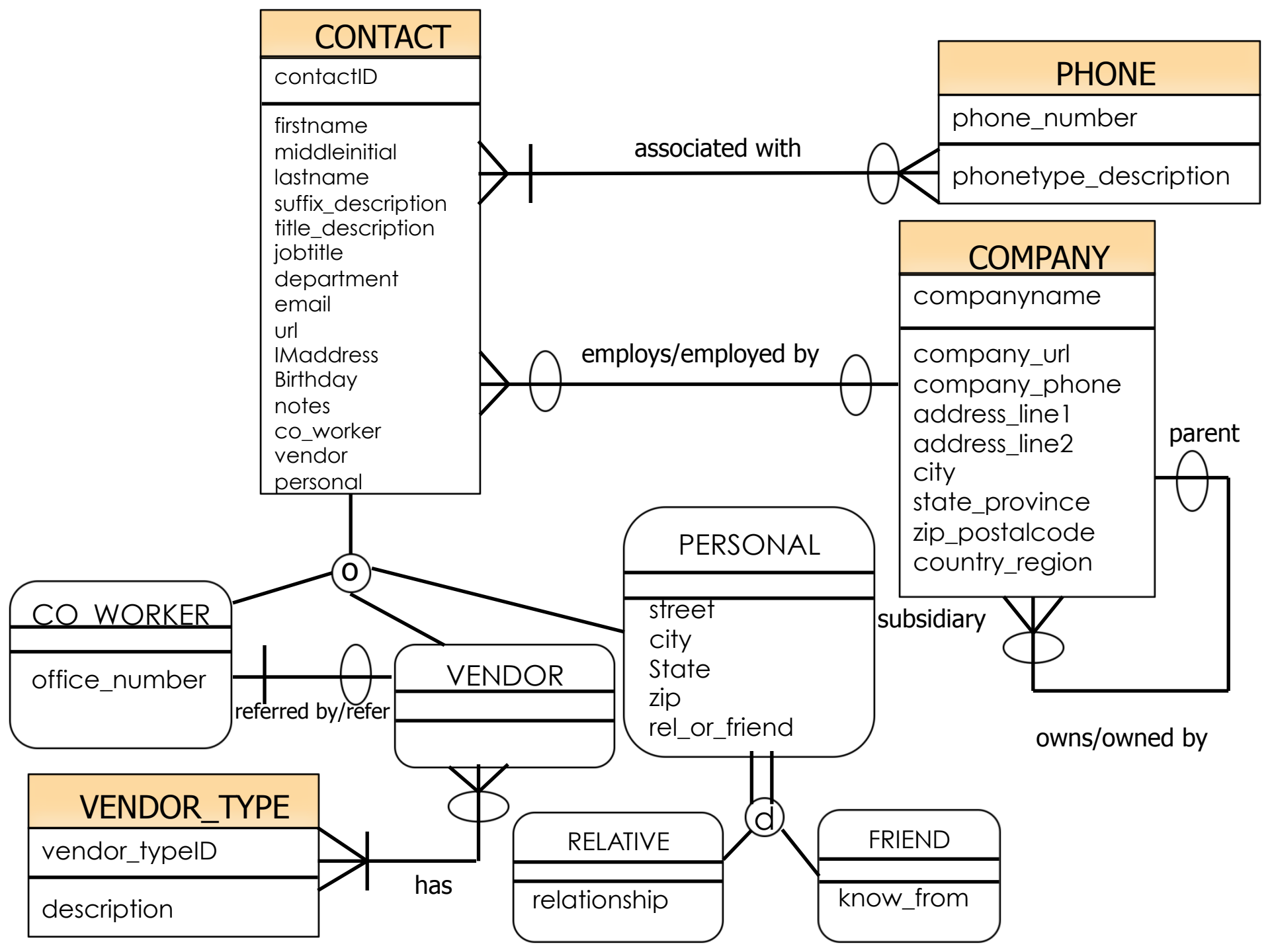
REPRESENTATIONAL level
(Relational Model)

DBMS software
which (to access the data), uses
the ...

DBMS

LOW-LEVEL or *physical* level
(e.g., file layouts/structures,
indexing, OS access strategies)




CONTACT
contactID
firstname
middleinitial
lastname
suffix_description
title_description
jobtitle
department
email
url
IMaddress
Birthday
notes
co_worker
vendor
personal
PHONE
phone_number
phonetype_description
associated with
COMPANY
companyname
company_url
company_phone
address_line1
address_line2
city
state_province
zip_postalcode
country_region
employs/employed by
parent
subsidiary
owns/owned by
PERSONAL
street
city
State
zip
rel_or_friend
o
CO_WORKER
office_number
referred by/refer
VENDOR
VENDOR_TYPE
vendor_typeID
description
has
d
RELATIVE
relationship
FRIEND
know_from

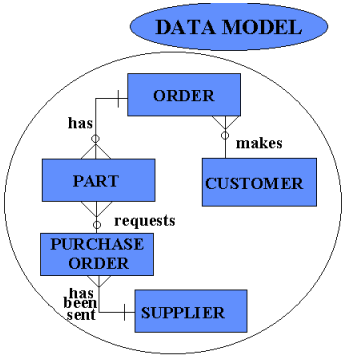
Expanded Database Development Lifecycle: CONMAN_II

- Summary of Steps
  1. Create a Data Model from gathered requirements.
     - Entity-Relationship Model
  2. **Transpose Data Model into Relation(s)**
  3. Normalize Relations
     A. Determine functional dependencies
     B. Determine candidate key(s)
     C. Confirm or modify "proposed" primary key
     D. Complete Normalization Process from First Normal Form (1NF) through Boyce-Codd Normal Form (BCNF) (per Normalization Process Reference sheet)
  4. Create the Relational Schema within Database Management System. (Metadata)
     - Create the database and the table(s)
  5. Define forms, queries, reports, menus, (Application Metadata), if supported within DBMS or external application programs.
  6. Populate database with User Data
  7. Maintenance

Review:

UoD

HIGH-LEVEL or *conceptual* level
(e.g., E-R diagram)



DBMS - INDEPENDENT

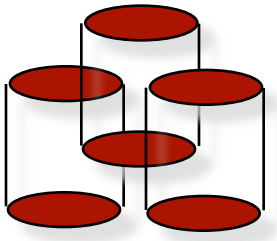
DBMS - SPECIFIC

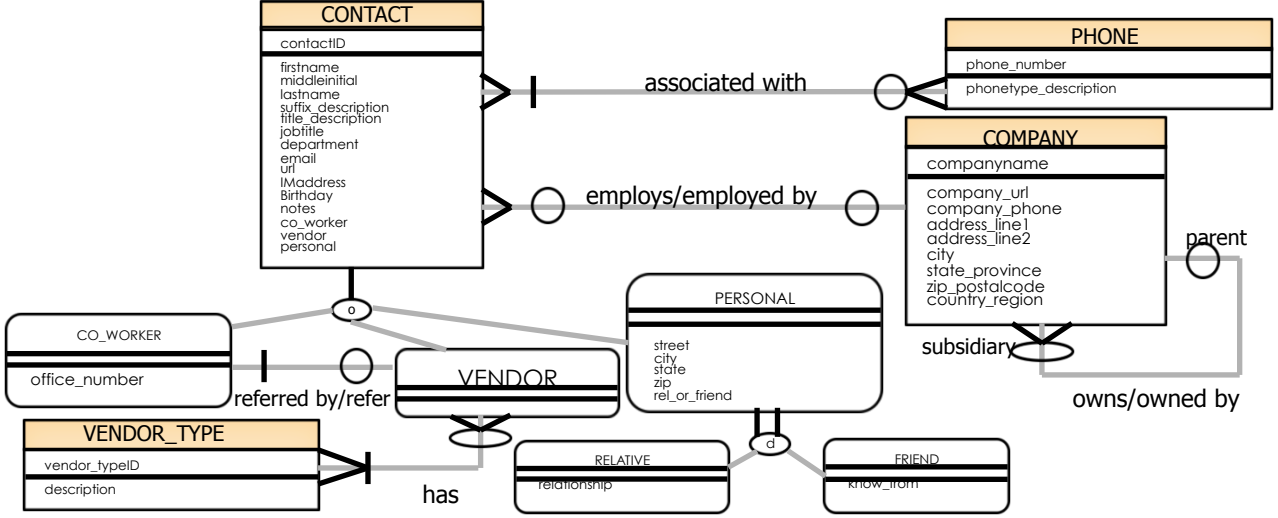
REPRESENTATIONAL level
(Relational Model)

DBMS software
which (to access the data), uses
the ...

DBMS

LOW-LEVEL or *physical* level
(e.g., file layouts/structures,
indexing, OS access strategies)





CONTACT(<u>contactID</u>, firstname, middleinitial, lastname, suffix_description, title_description, jobtitle, department, email, url, IMaddress, birthday, notes, co_worker, vendor, personal, *companyname*)
CONTACT(companyname) mei COMPANY(companyname)

PHONE(<u>phone_number</u>, phonetype_description)

COMPANY(<u>companyname</u>, company_url, company_phone, address_line1, address_line2, city, state_province, zip_postalcode, country_region, *parent_companyname*)
COMPANY(parent_companyname) mei COMPANY(companyname)

CO_WORKER(<u>*contactID*</u>, office_number)
CO_WORKER(contactID) mei CONTACT(contactID)

VENDOR(<u>*contactID*</u>, *co_worker_referrer_contactID*)
VENDOR(contactID) mei CONTACT(contactID)
VENDOR(co_worker_referrer_contactID) mei CO_WORKER(contactID)

PERSONAL(<u>*contactID*</u>, street, city, state, zip, rel_or_friend)
PERSONAL(contactID) mei CONTACT(contactID)

VENDOR_TYPE(<u>vendor_typeID</u>, description)

RELATIVE(<u>*contactID*</u>, relationship)
RELATIVE(contactID) mei PERSONAL(contactID)

FRIEND(<u>*contactID*</u>, know_from)
FRIEND(contactID) mei PERSONAL(contactID)

VENDOR_VENDOR_TYPE(<u>*contactID, vendor_typeID*</u>)
VENDOR_VENDOR_TYPE(contactID) mei VENDOR(contactID)
VENDOR_VENDOR_TYPE(vendor_typeID) mei VENDOR_TYPE(vendor_typeID)

CONTACT_PHONE(<u>*contactID, phone_number*</u>)
CONTACT_PHONE(contactID) mei CONTACT(contactID)
CONTACT_PHONE(phone_number) mei PHONE(phone_number)

Expanded Database Development Lifecycle: CONMAN_II

- Summary of Steps
  1. Create a Data Model from gathered requirements.
     - Entity-Relationship Model
  2. Transpose Data Model into Relation(s)
  3. **Normalize Relations**
     A. Determine functional dependencies
     B. Determine candidate key(s)
     C. Confirm or modify "proposed" primary key
     D. Complete Normalization Process from First Normal Form (1NF) through Boyce-Codd Normal Form (BCNF) (per Normalization Process Reference sheet)
  4. Create the Relational Schema within Database Management System. (Metadata)
     - Create the database and the table(s)
  5. Define forms, queries, reports, menus, (Application Metadata), if supported within DBMS or external application programs.
  6. Populate database with User Data
  7. Maintenance

Normalizing ConMan II

CONTACT(contactID, firstname, middleinitial, lastname, suffix_description, title_description, jobtitle, department, email, url, IMaddress, birthday, notes, co_worker, vendor, personal, *companyname*)

CONTACT(companyname) mei COMPANY(companyname)

FDs:

contactID➔firstname, middleinitial, lastname, suffix_description, title_description, jobtitle, department, email, url, IMaddress, birthday, notes, co_worker, vendor, personal, *companyname*

email ➔contactID, firstname, middleinitial, lastname, suffix_description, title_description, jobtitle, department, url, Imaddress, birthday, notes, co_worker, vendor, personal, *companyname*

PHONE(phone_number, phonetype_description)

FDs:

phone_number ➔phonetype_description

COMPANY(companyname, company_url, company_phone, address_line1, address_line2, city, state_province, zip_postalcode, country_region, *parent_companyname*)

COMPANY(parent_companyname) mei COMPANY(companyname)

FDs:

companyname➔company_url, company_phone, address_line1, address_line2, city, state_province, zip_postalcode, country_region, *parent_companyname*

company_url➔companyname, company_phone, address_line1, address_line2, city, state_province, zip_postalcode, country_region, *parent_companyname*

company_phone➔companyname, company_url, address_line1, address_line2, city, state_province, zip_postalcode, country_region, *parent_companyname*

address_line1, address_line2, zip_postalcode ➔companyname, company_url, company_phone, city, state_province, country_region,*parent_companyname*

Normalizing ConMan II con't

CO_WORKER(*contactID,* office_number)
CO_WORKER(contactID) mei CONTACT(contactID)
FDs:
contactID ➔office_number

VENDOR(*contactID*, *co_worker_referrer_contactID*)
VENDOR(contactID) mei CONTACT(contactID)
VENDOR(co_worker_referrer_contactID) mei CO_WORKER(contactID)
FDs:
contactID ➔*co_worker_referrer_contactID*

PERSONAL(*contactID*, street, city, state, zip, rel_or_friend)
PERSONAL(contactID) mei CONTACT(contactID)
FDs:
contactID ➔street, city, state, zip, rel_or_friend

VENDOR_TYPE(vendor_typeID, description)
FDs:
vendor_typeID ➔ description

RELATIVE(*contactID*, relationship)
RELATIVE(contactID) mei PERSONAL(contactID)
FDs:
contactID ➔ relationship

Normalizing ConMan II con't

FRIEND(*contactID*, know_from)
FRIEND(contactID) mei PERSONAL(contactID)

FDs:
contactID ➔know_from

VENDOR_VENDOR_TYPE(*contactID, vendor_typeID*)
VENDOR_VENDOR_TYPE(contactID) mei VENDOR(contactID)
VENDOR_VENDOR_TYPE(vendor_typeID) mei VENDOR_TYPE(vendor_typeID)

CONTACT_PHONE(*contactID, phone_number*)
CONTACT_PHONE(contactID) mei CONTACT(contactID)
CONTACT_PHONE(phone_number) mei PHONE(phone_number)

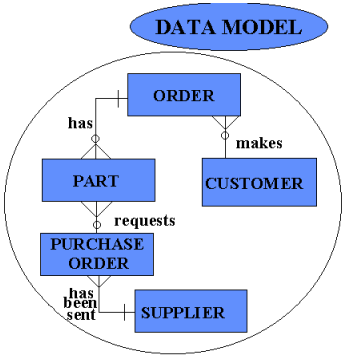
Expanded Database Development Lifecycle: CONMAN_II

- Summary of Steps
  1. Create a Data Model from gathered requirements.
     - Entity-Relationship Model
  2. Transpose Data Model into Relation(s)
  3. Normalize Relations
  4. **Create the Relational Schema within Database Management System. (Metadata)**
     a. Create the database
     b. Create the table(s)
        i. Order matters – start with tables that do not have any foreign keys
  5. Define forms, queries, reports, menus, (Application Metadata), if supported within DBMS or external application programs.
  6. Populate database with User Data
  7. Maintenance

Review:

UoD

HIGH-LEVEL or *conceptual* level
(e.g., E-R diagram)



DBMS - INDEPENDENT

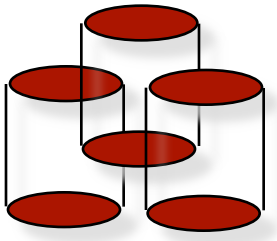
DBMS - SPECIFIC

REPRESENTATIONAL level
(Relational Model)

DBMS software
which (to access the data), uses
the ...

DBMS

LOW-LEVEL or *physical* level
(e.g., file layouts/structures,
indexing, OS access strategies)



What order do we create the tables in?

CONTACT(contactID, firstname, middleinitial, lastname, suffix_description, title_description, jobtitle, department, email, url, IMaddress, birthday, notes, co_worker, vendor, personal, *companyname*)
CONTACT(companyname) mei COMPANY(companyname)

PHONE(phone_number, phonetype_description)

COMPANY(companyname, company_url, company_phone, address_line1, address_line2, city, state_province, zip_postalcode, country_region, *parent_companyname*)
COMPANY(parent_companyname) mei COMPANY(companyname) **(Implementing recursive foreign key (slide 18))**

CO_WORKER(*contactID,* office_number) **(Adding Data (slide 19))**
CO_WORKER(contactID) mei CONTACT(contactID)

VENDOR(*contactID*, *co_worker_referrer_contactID*)
VENDOR(contactID) mei CONTACT(contactID) **(Implementing foreign keys for supertype/subtypes (slide 18))**
VENDOR(co_worker_referrer_contactID) mei CO_WORKER(contactID)

PERSONAL(*contactID*, street, city, state, zip, rel_or_friend)
PERSONAL(contactID) mei CONTACT(contactID)

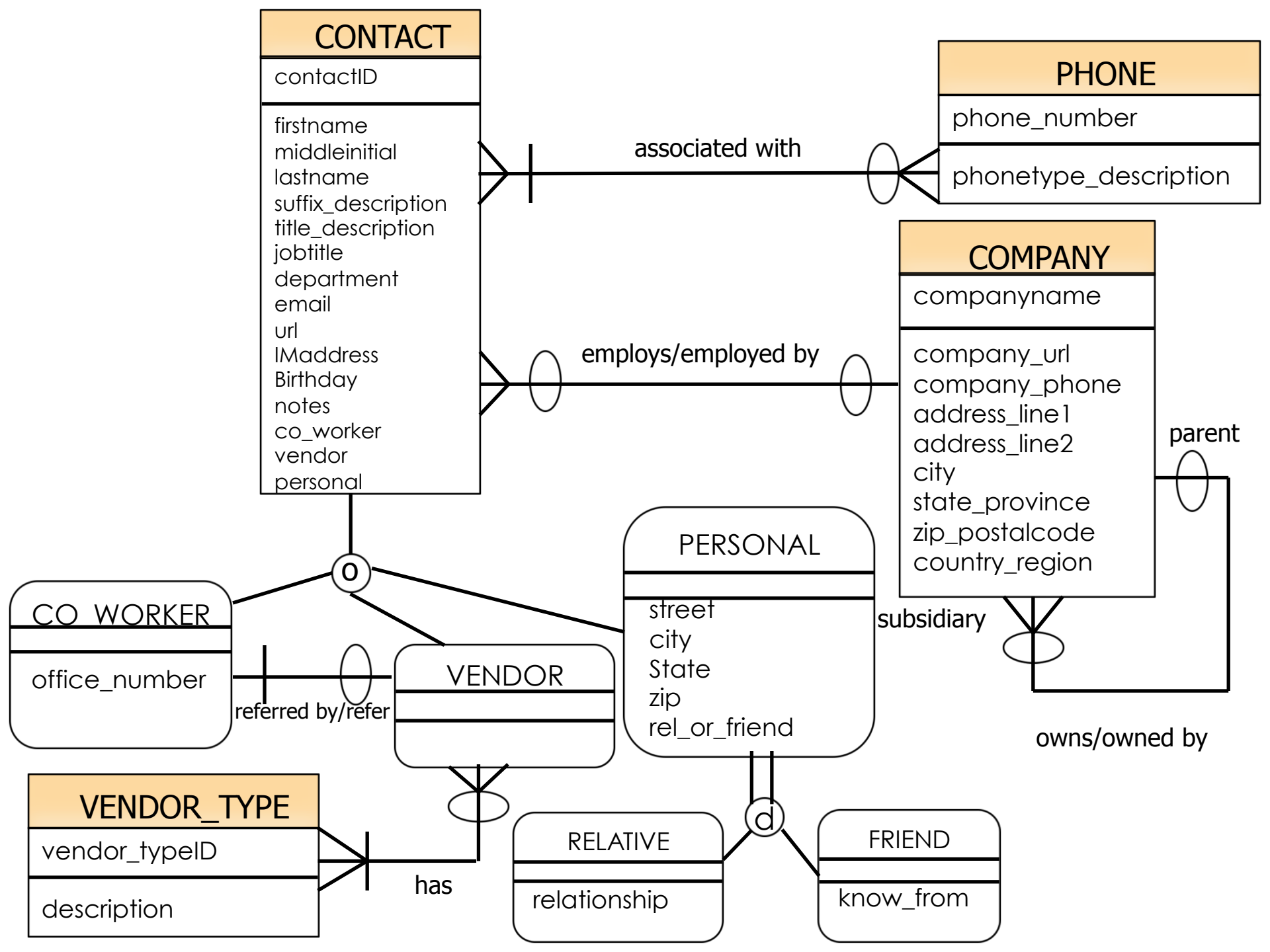
VENDOR_TYPE(vendor_typeID, description) **(lookup tables (slide 16); additional data types (slide 17)**

RELATIVE(*contactID*, relationship)
RELATIVE(contactID) mei PERSONAL(contactID)

FRIEND(*contactID*, know_from)
FRIEND(contactID) mei PERSONAL(contactID)

VENDOR_VENDOR_TYPE(*contactID, vendor_typeID*)
VENDOR_VENDOR_TYPE(contactID) mei VENDOR(contactID)
VENDOR_VENDOR_TYPE(vendor_typeID) mei VENDOR_TYPE(vendor_typeID)

CONTACT_PHONE(*contactID, phone_number*)
CONTACT_PHONE(contactID) mei CONTACT(contactID)
CONTACT_PHONE(phone_number) mei PHONE(phone_number)


CONTACT
contactID
firstname
middleinitial
lastname
suffix_description
title_description
jobtitle
department
email
url
IMaddress
Birthday
notes
co_worker
vendor
personal
PHONE
phone_number
phonetype_description
associated with
COMPANY
companyname
company_url
company_phone
address_line1
address_line2
city
state_province
zip_postalcode
country_region
employs/employed by
parent
subsidiary
owns/owned by
PERSONAL
street
city
State
zip
rel_or_friend
o
CO_WORKER
office_number
referred by/refer
VENDOR
VENDOR_TYPE
vendor_typeID
description
has
d
RELATIVE
relationship
FRIEND
know_from

Lookup Table

- Also known as Reference table
- Used to minimize storage of repetitive values
- Ensures sure that only certain values are allowed
  - Helpful to create drop-down list of options in front-end interface
- Typical use:
  - Store a code in the "main" table
  - Store the code and full value in the lookup table
- Example: VENDOR_TYPE
- Example:

Address

Name	Street	City	State
Fred	123 Main St	Rochester	NY
Sally	45 Elm St	Auburn	MA

State in Address references State in StateCode

StateCode

State	StateName
CT	Connecticut
MA	Massachusetts
NY	New York

Additional Data Types

- TINYINT **(createVENDOR_TYPE.sql)**
  - http://dev.mysql.com/doc/refman/5.6/en/integer-types.html
  - UNSIGNED **(createVENDOR_TYPE.sql)** – only holds positive values
- BOOLEAN (BOOL)
  - Not directly implemented in MySQL, instead TINYINT(1); 0=False; 1=True (createCONTACT.sql)
- ENUM
  - Allows a fixed number of pre-defined values to be specified (createPERSONAL.sql)
- Data Type Attribute:
  - AUTO_INCREMENT
    - Can be used with whole number data types to automatically supply a value incremented by 1, for each record inserted.
    - To utilize, include in attribute specification, then don't supply a value for the respective attribute when inserting new records
      - 0 or NULL will also cause an automatic value to be used for the respective attribute
    - Example: **createVENDOR_TYPE.sql**

Implementing a Foreign Key

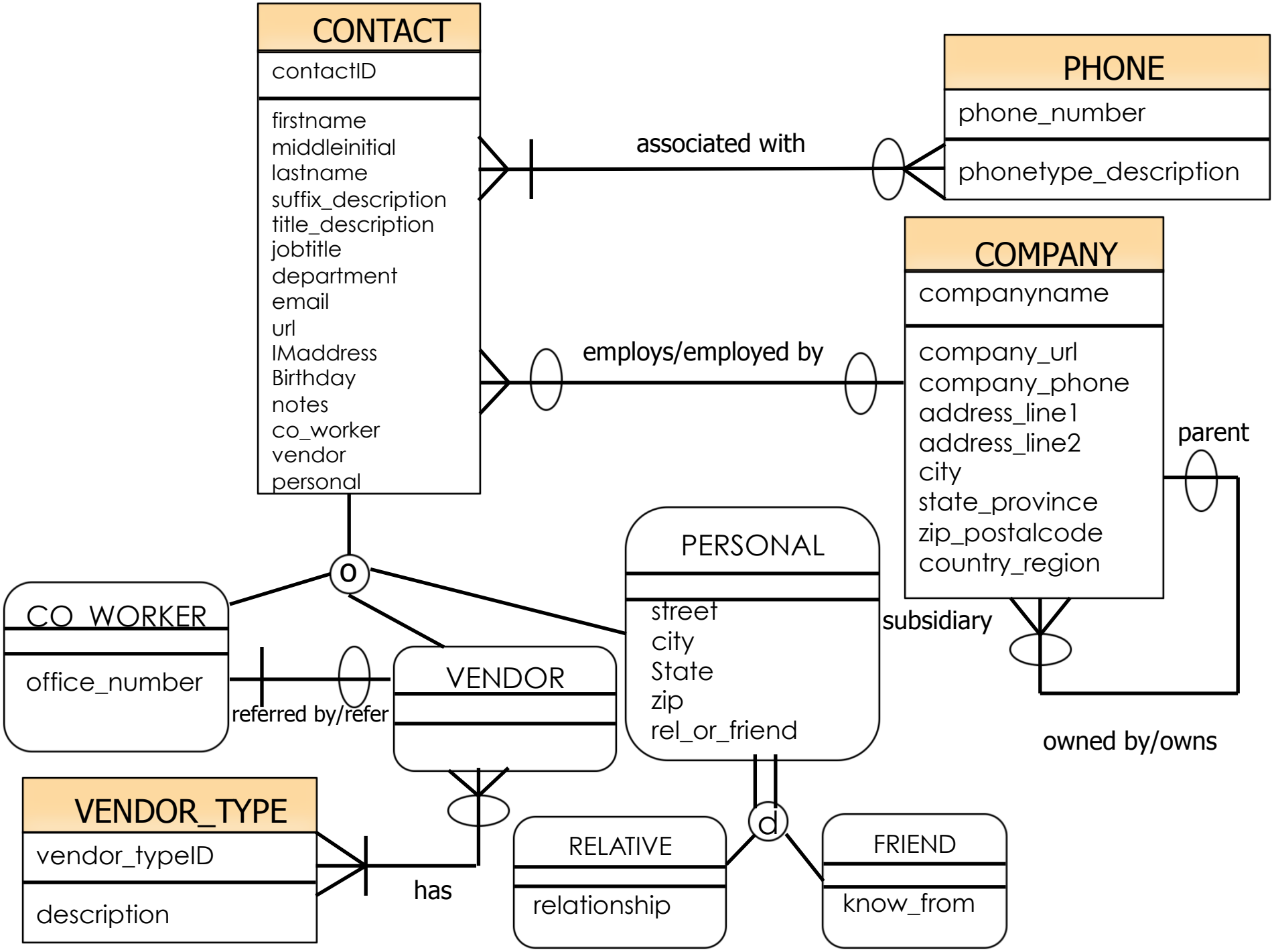
- Enforces **referential integrity**: can't add a value to a foreign key, unless that value exists for the foreign key back in it's "native" table.
- Two step process in a CREATE TABLE statment:
  - Add an attribute speficiation for each foreign key attribute
    - Datatype must be consistent
    - Refer to E-R diagram to determine in a NOT NULL constraint is needed for the foreign key attribute(s)
  - Add a **foreign key constraint** for each foreign key
    - Syntax:
      - CONSTRAINT *table_attr(s)_fk* FOREIGN KEY (*attr(s)*) REFERENCES *native_table*(*attr(s)*)
- For 1:1 or 1:N (N:1) recursive relationships (createCOMPANY.sql)
  - Foreign key constraint will reference the same table's primary key
- For supertype/subtype relationships (createVENDOR.sql)
  - Foreign key constraint in subtype will reference supertype

Adding Data

- INSERT using value(s) from another table (createCO_WORKER.sql)
  - Instead of looking up values needed to compose an INSERT statement, include a query that will find and use the needed values.
  - Example:
    - If we want to add a instance of a subtype, we need to make sure that the primary key values match, therefore finding the value to use from the supertype is necessary. In this case, what contactID do we need to use to have William Destler (CONTACT) as a CO_WORKER?
      - SELECT contactID FROM CONTACT WHERE firstname='William' and lastname='Destler';
      - INSERT INTO co_worker (contactID, office_number) VALUES(1, 'Bldg. 1, 7th Floor');
    - **INSERT INTO co_worker (contactID, office_number) SELECT contactID, 'Bldg. 1, 7th Floor' FROM contact WHERE firstname='William' and lastname='Destler';**

New files

- createCONMAN_DB_v2FULL.sql
  - Includes all CREATE statements to implement CONMAN v2.
  - Reminder: In order to determine if a NOT NULL constraint is needed for a foreign key attribute, we need to refer to the E-R diagram.
- insertDestler.sql
  - Includes all INSERT and UPDATE statements needed to add Destler's information into the v2 structure (more than one table).


CONTACT
contactID
firstname
middleinitial
lastname
suffix_description
title_description
jobtitle
department
email
url
IMaddress
Birthday
notes
co_worker
vendor
personal
associated with
PHONE
phone_number
phonetype_description
employs/employed by
COMPANY
companyname
company_url
company_phone
address_line1
address_line2
city
state_province
zip_postalcode
country_region
parent
subsidiary
owned by/owns
PERSONAL
street
city
State
zip
rel_or_friend
o
CO_WORKER
office_number
referred by/refer
VENDOR
VENDOR_TYPE
vendor_typeID
description
has
d
RELATIVE
relationship
FRIEND
know_from

A closer look at insertDestler.sql

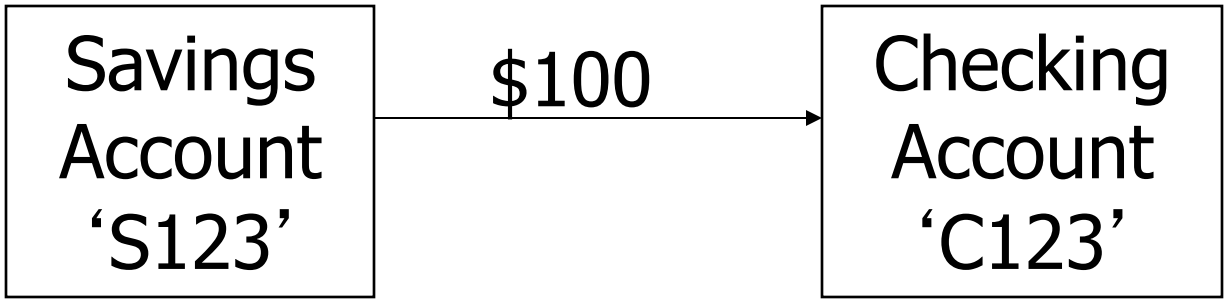
- Now that our database schema has more than one table, it is now necessary to have multiple statements to add the needed data.
- Review: INSERT for CO_WORKER adds values based on a SELECT statement
  - Saves time looking up needed values (contactID) from another table
- Note that it takes two statements to add Destler to a subtype:
  - INSERT – to add data to the subtype
  - UPDATE – to change discriminator(s) appropriately
  - We need to ensure that both statements will execute successfully or not at all, otherwise, the data will be inconsistent (data in subtype, but discriminator not reflective or vice versa, discriminator gets updated, but subtype record not added)
    - Either case is BAD! Data is not complete and left in an inconsistent state
    - How can this situation be prevented?

Transactions

- SQL provides us with the concept of a "transaction".
- A transaction is a logical unit of work.
- Can be composed of one or more statements
- Statements can be
  - SQL
    - Data Definition Language (DDL)
    - Data Manipulation Language (DML)
  - Programming elements - if supported by DBMS
- Processed by Transaction Control Language (TCL) statements
  - COMMIT = save
  - ROLLBACK = undo/cancel

Bank Account Example

- Imagine you have a Savings Account and a Checking Account
  - The savings account pays interest so you keep money there
  - The checking account is for paying bills
- Let's say you want to move $100 from your savings account to your checking account



- The bank uses a database to maintain your account data
- What needs to happen for this transfer to occur?

Bank Account Example

- First, we need to debit (remove) the money from your savings account:

```
UPDATE Savings

SET Balance = Balance - 100

WHERE Account = 'S123';
```

- Next, we need to credit (add) the money to your checking account:

```
UPDATE Checking

SET Balance = Balance + 100

WHERE Account = 'C123';
```

Bank Account Trouble

- That would work nicely, MOST OF THE TIME!
- What if something happened in between those two SQL statements
  - Disk crash
  - Power failure
  - DBMS stops responding unexpectedly
  - Or a variety of other things
- Suddenly, we have a problem

All or Nothing

- Many situations require a group of SQL statements to be executed in their entirety, or not at all
- "Inventory" related domains commonly fall into this category
  - Transfer of goods
  - Transfer of money

Properties of a Transaction (ACID)

- Atomicity
  - A transaction is considered an atomic unit. A transaction is resolved by either all changes being made or none of the changes being made, a partial commit is not allowed.
  - If a failure occurs in the middle of a transaction, all partial results must be undone.
- Consistency
  - A transaction must leave the database in a consistent state.
  - A transaction must not violate database integrity constraints.
- Isolation
  - Concurrent transactions will not negatively impact one another.
    - Record locking – can restrict access to records that are part of an in-process transaction.
  - An in-process transaction will not reveal results to others until it is committed or rolled-back.
- Durability
  - Committed data is permanently recorded.
  - If a failure occurs, DBMS, needs to be able to restore committed data.

Transactions in MySQL

- Default behavior is that each statement is a "transaction".
  - Can temporarily override by START TRANSACTION
  - Can disable for current session by SET AUTOCOMMIT=0;
  - http://dev.mysql.com/doc/refman/5.7/en/commit.html

```
START TRANSACTION;
   UPDATE Savings
         SET Balance = Balance - 100
         WHERE Account = 'S123';
   UPDATE Checking
         SET Balance = Balance + 100
         WHERE Account = 'C123';
COMMIT;
```

COMMIT

- The COMMIT statement defines the end of the transaction and writes the changes to the database
- In MySQL, an implicit commit is performed when a DDL statement is excuted with an in-process transaction
  - http://dev.mysql.com/doc/refman/5.7/en/implicit-commit.html
- The ACID properties guarantee that all the changes between the "BEGIN" and "COMMIT" statements get done or not done
- The database is automatically restored to its prior state if there is a problem.
  - No code or other intervention by programmer

Review

- A transaction is a unit of work
- The DBMS guarantees that the transaction will have the ACID properties
- Many DBMS packages require a "START TRANSACTION" statement to signify the start of the unit of work
- The COMMIT statement defines the end of the transaction and writes the changes to the database if and only if there were no non-trappable problems
- The ROLLBACK statement restores the database to its state before the transaction began

What if…

- we wanted to remove William Destler as a contact?

```
DELETE FROM contact
WHERE contactID=1;
```

  - What happens?
    - ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails (`conmanv2`.`co_worker`, CONSTRAINT `co_worker_contactID_fk` FOREIGN KEY (`contactID`) REFERENCES `contact` (`contactID`))What does this even mean?
      - We can't delete Destler as a CONTACT because we also have him included as a CO_WORKER (subtype of CONTACT). If we delete from CONTACT, then we have a subtype with no related supertype = bad!

Option 1

- Delete Destler's record from CO_WORKER, then delete his record from CONTACT.

```
DELETE FROM co_worker
WHERE contactID=1;
DELETE FROM contact
WHERE contactID=1;
```

  - What happened?
    - ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails (`conmanv2`.`personal`, CONSTRAINT `personal_contactID_fk` FOREIGN KEY (`contactID`) REFERENCES `contact` (`contactID`))
      - OK, so we need to delete his record from PERSONAL
        - What happened?
          - ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails (`conmanv2`.`friend`, CONSTRAINT `friend_contactID_fk` FOREIGN KEY (`contactID`) REFERENCES `personal` (`contactID`))
            - OK, so we need to delete his record from Friend
              - What happened? It worked!
              - Now we can delete his record from PERSONAL, then delete his record from CONTACT?
                - What happened?

Option 1 con't

- Recap: Now we can delete his record from PERSONAL, then delete his record from CONTACT?
  - What happened?
    - Delete from PERSONAL worked
    - Delete from CONTACT = ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails (`conmanv2`.`contact_phone`, CONSTRAINT `contact_phone_contactID_fk` FOREIGN KEY (`contactID`) REFERENCES `contact` (`contactID`))
      - So now we need to delete Destlers record from CONTACT_PHONE.
        - It worked
      - Now can we delete Destler from CONTACT?
        - Yes, finally! That took WAY TOO MUCH WORK!
- We want our foreign keys to be enforced so that our data is consistent, but how do we prevent this amount of work just to remove a contact?

Option 2: Foreign Key specifications

- FOREIGN KEYS
  - ON UPDATE and/or ON DELETE
    - CASCADE
    - SET NULL
    - SET DEFAULT
    - NO ACTION
  - Example:
    - createCONMAN_DB_v3.sql –view to see how ON UPDATE and ON DELETE was used.
    - COMMITinsertDestler.sql
    - updateDestler.sql
    - deleteDestler.sql

ALTER TABLE: ADD/DROP FOREIGN KEY CONSTRAINT

- You can't modify a FOREIGN KEY constraint; it must be dropped and added.

```
ALTER TABLE tablename
DROP FOREIGN KEY constraint_name_fk;


ALTER TABLE tablename
ADD CONSTRAINT constraint_name_fk FOREIGN KEY (attr(s))
REFERENCES native_table_name (attr(s)
        [ON DELETE option]
        [ON UPDATE option];
```