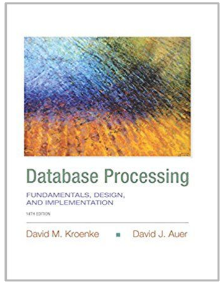
Fairfield University DOLAN School of Business


Database Processing
FUNDAMENTALS, DESIGN, AND IMPLEMENTATION
David M. Kroenke · David J. Auer

Databases for Analytics

Kroenke / Auer
Chapters 6 and 7 (partial)

Learning Objectives

- **Skills:** You should know how to ...
  - Prepare ERDs for conversion to SQL DDL
  - Write SQL DDL code to create, modify, and drop tables
  - Identify various types of SQL DDL statements
- **Theory:** You should be able to explain ...
  - Why and how many-to-many relationships must be converted to (pairs of) one-to-many relationships
  - The use of constraints and triggers to avoid referential integrity violations

ERDs to Table Designs

A little planning before you write SQL

The Basic Process

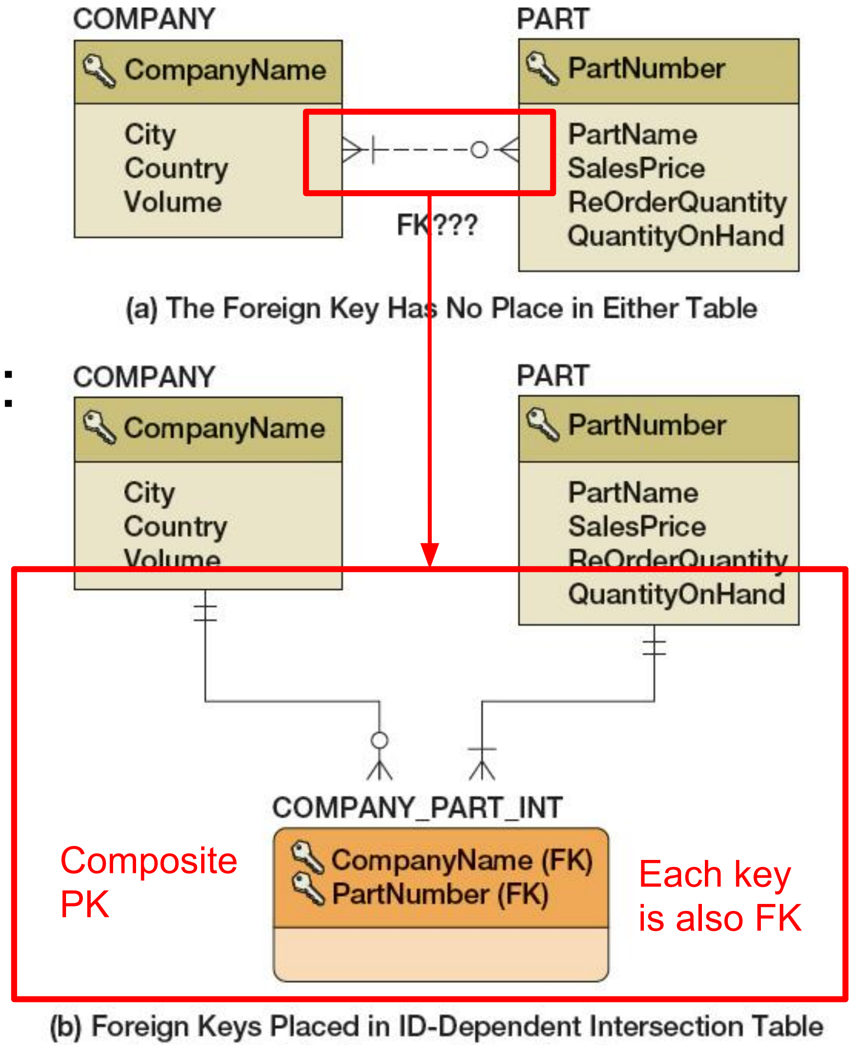
1. **Convert many-to-many relationships to intersection tables/associative entities.**
2. **Define a table for each entity class.**
    - Use ALL_CAPS and underscores for table names.
3. **Define a column for each attribute.**
    - Use CamelCase for the attribute names (with no spaces).
    - Select data types based on the attribute domain.
4. **Define PK and FK columns as needed.**
    - Common practice is to create surrogate PKs, then define FK columns to match. Every *many* cardinality → FK field.
5. **Define NULL statuses, DEFAULTs, and Constraints.**

Step 1. Eliminate N:M Relationships

Two logically equivalent cases:

- "Intersection table" entity (like at lower right) has two FKs and no attributes.
- Associative entity has no attributes (but we can add them later)

So, an N:M is converted to an entity (table) with two 1:N relationships (with FKs that are also PK).



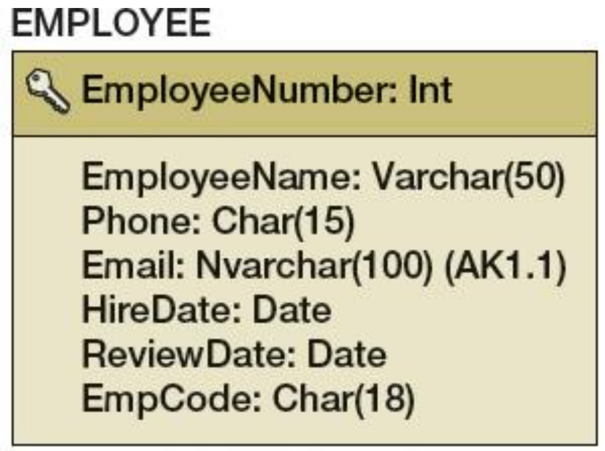
(a) The Foreign Key Has No Place in Either Table

(b) Foreign Keys Placed in ID-Dependent Intersection Table

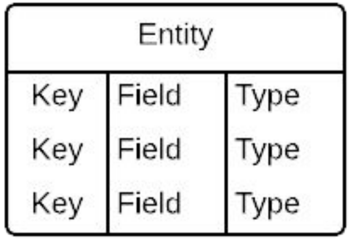
Steps 2 and 3. Tables and Attributes

In practice this just means **adding data types** and other field properties to your ERDs.

Each field then represents a table column.


EMPLOYEE

EmployeeNumber: Int
EmployeeName: Varchar(50) Phone: Char(15) Email: Nvarchar(100) (AK1.1) HireDate: Date ReviewDate: Date EmpCode: Char(18)

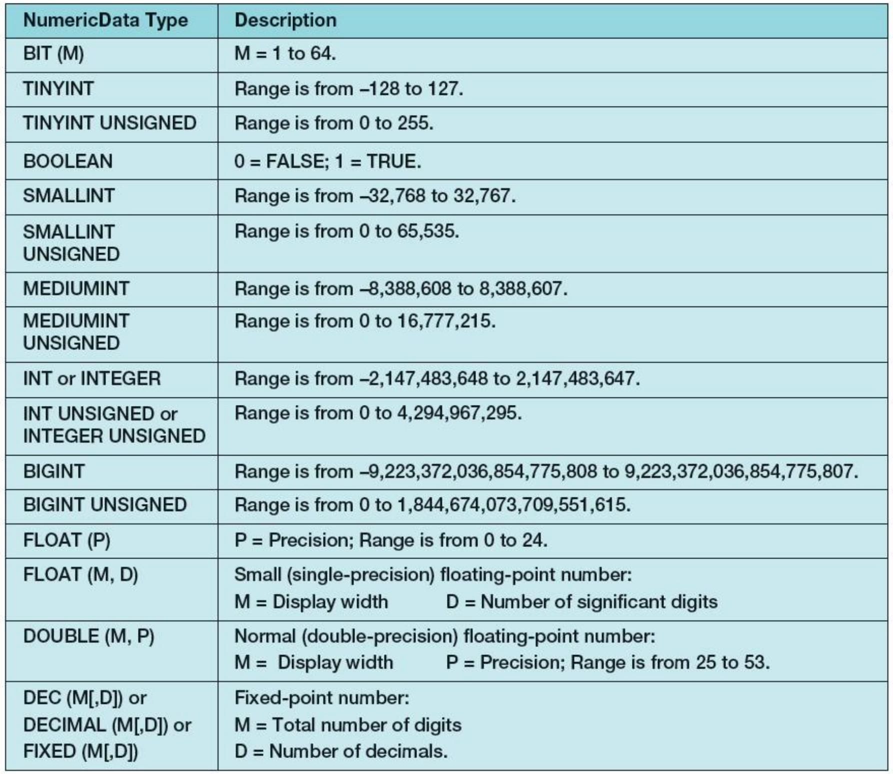

Entity

Key	Field	Type
Key	Field	Type
Key	Field	Type

A more "correct" entity template

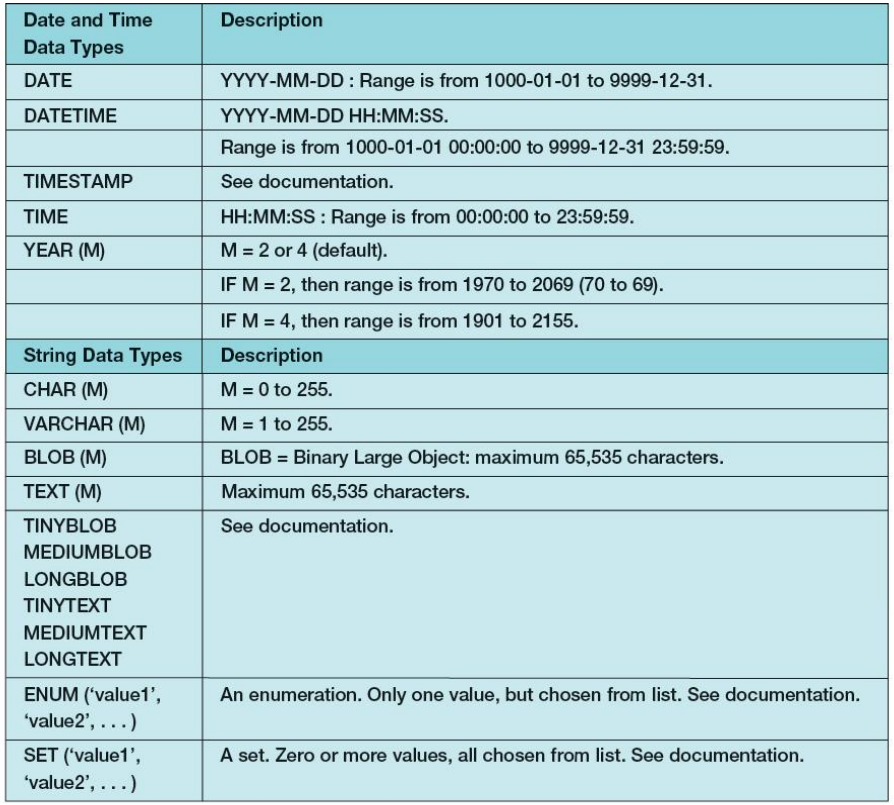
Data Types Options

One can usually get along with just INTEGER, FLOAT, VARCHAR(n), and TEXT



NumericData Type	Description
BIT (M)	M = 1 to 64.
TINYINT	Range is from –128 to 127.
TINYINT UNSIGNED	Range is from 0 to 255.
BOOLEAN	0 = FALSE; 1 = TRUE.
SMALLINT	Range is from –32,768 to 32,767.
SMALLINT UNSIGNED	Range is from 0 to 65,535.
MEDIUMINT	Range is from –8,388,608 to 8,388,607.
MEDIUMINT UNSIGNED	Range is from 0 to 16,777,215.
INT or INTEGER	Range is from –2,147,483,648 to 2,147,483,647.
INT UNSIGNED or INTEGER UNSIGNED	Range is from 0 to 4,294,967,295.
BIGINT	Range is from –9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.
BIGINT UNSIGNED	Range is from 0 to 1,844,674,073,709,551,615.
FLOAT (P)	P = Precision; Range is from 0 to 24.
FLOAT (M, D)	Small (single-precision) floating-point number: M = Display width D = Number of significant digits
DOUBLE (M, P)	Normal (double-precision) floating-point number: M = Display width P = Precision; Range is from 25 to 53.
DEC (M[,D]) or DECIMAL (M[,D]) or FIXED (M[,D])	Fixed-point number: M = Total number of digits D = Number of decimals.

(c) Common Data Types in MySQL 5.6



Date and Time Data Types	Description
DATE	YYYY-MM-DD : Range is from 1000-01-01 to 9999-12-31.
DATETIME	YYYY-MM-DD HH:MM:SS.
	Range is from 1000-01-01 00:00:00 to 9999-12-31 23:59:59.
TIMESTAMP	See documentation.
TIME	HH:MM:SS : Range is from 00:00:00 to 23:59:59.
YEAR (M)	M = 2 or 4 (default).
	IF M = 2, then range is from 1970 to 2069 (70 to 69).
	IF M = 4, then range is from 1901 to 2155.
String Data Types	**Description**
CHAR (M)	M = 0 to 255.
VARCHAR (M)	M = 1 to 255.
BLOB (M)	BLOB = Binary Large Object: maximum 65,535 characters.
TEXT (M)	Maximum 65,535 characters.
TINYBLOB MEDIUMBLOB LONGBLOB TINYTEXT MEDIUMTEXT LONGTEXT	See documentation.
ENUM (‘value1’, ‘value2’, . . .)	An enumeration. Only one value, but chosen from list. See documentation.
SET (‘value1’, ‘value2’, . . .)	A set. Zero or more values, all chosen from list. See documentation.

(c) continued - Common Data Types in MySQL 5.6

Step 4. PK and FK fields

Make sure the tables are sufficiently normalized before finalizing your keys.

Primary Keys can be either native attributes or artificial surrogates (auto-numbered). Surrogates are the norm, however.

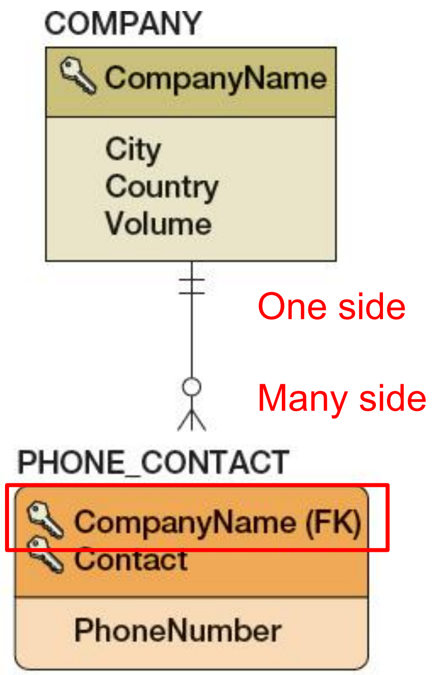
Foreign keys are both *fields* and *index constraints*. Define the FK field first (with a type that is compatible with the PK) and then create the FK constraint.

Every crows foot is an FK

Crows feet appear on the **many** side of a relationship, with the other end always the **one** side.

Since an FK field can only point to one record, it must point to the one side.

Thus, it is on the many side.



Step 5. Set Field Options/Constraints

- **Can the field allow NULL values?**
  - PK fields never allow NULLs.
  - FKs can allow NULLs if the relationship is optional
  - For other fields, use your judgment
- **Does the field have a DEFAULT value?**
  - This is an easy way to avoid NULLs
- **Are there any (other) data constraints?**
  - **Range** constraints: min and max values on attributes
  - **Intrarelation** constraints apply to columns within a table
  - **Interrelation** constraints apply between tables

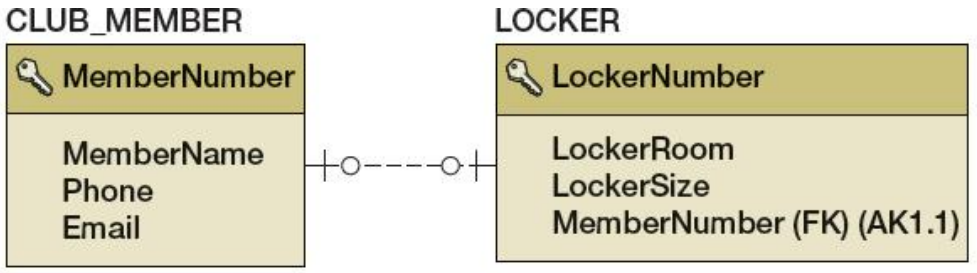
Special Cases

Patterns to follow when confronted with unusual situations

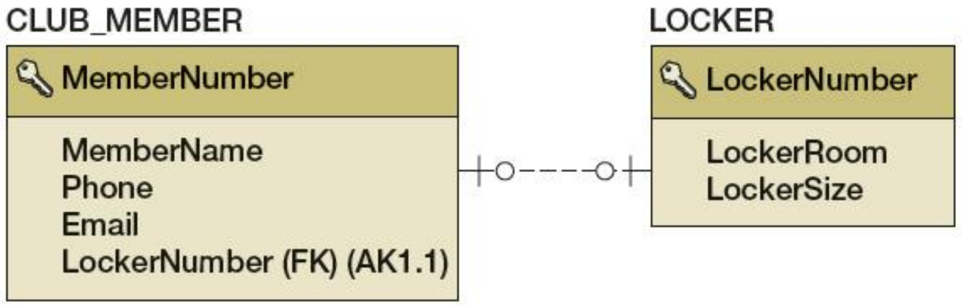
1:1 Relationships

Add an FK field ***opposite*** the **mandatory** side of the relationship.

If both ends are optional then pick either side, depending on data constraints.



(a) With Foreign Key in LOCKER

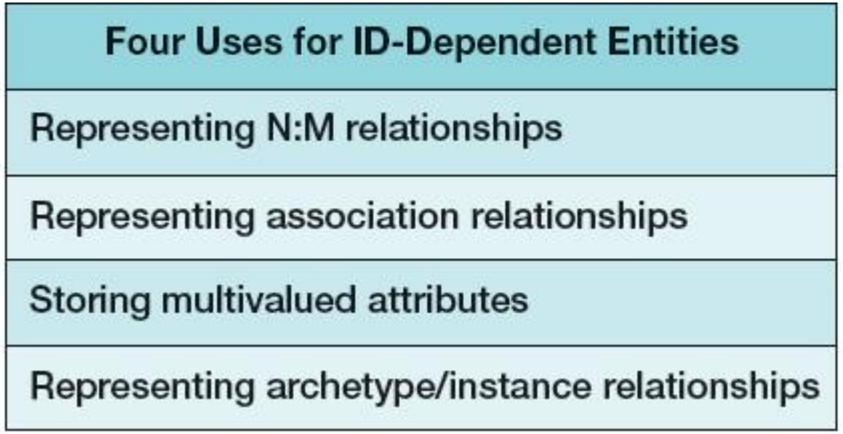


(b) With Foreign Key in CLUB_MEMBER

ID-Dependent (Weak) Entities

Always make the FK on the ID-Dependent side.

- Intersection tables and associative entities are always on FK side.
- Multivalued attributes are always on the FK side.
- Subtypes (children) are on the FK type opposite their supertypes (parents).

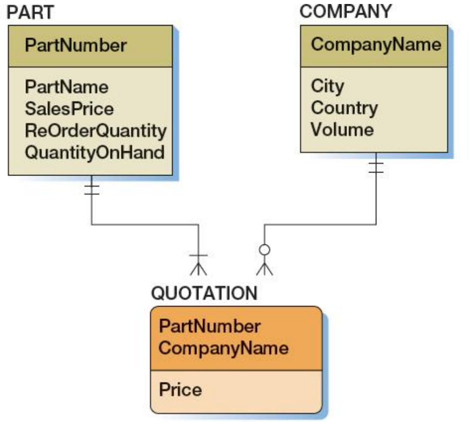


Four Uses for ID-Dependent Entities
Representing N:M relationships
Representing association relationships
Storing multivalued attributes
Representing archetype/instance relationships

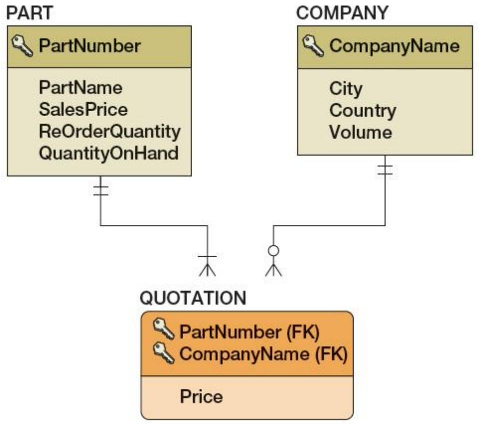
Intersection and Associative Entities

Note the crows feet on the ID-dependent entities.

That means they must have FKs that depend on the PKs of the related entities.

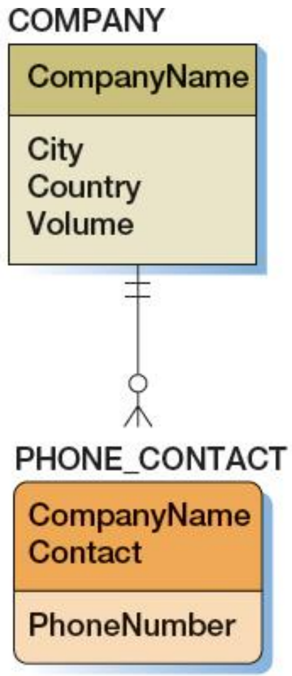


(a) Association Pattern Data Model from Figure 5-22

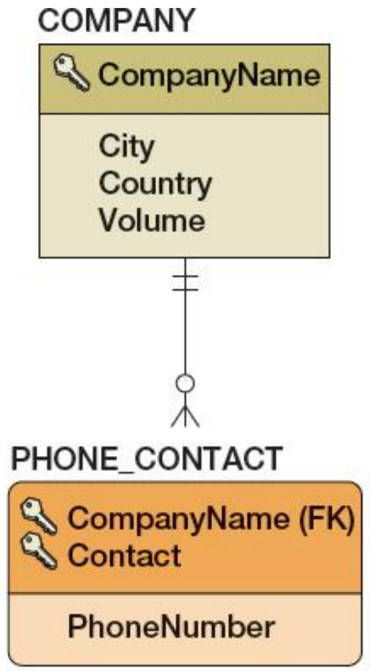


(b) Association Pattern Database Design

Multivalued Attributes ("repeating fields")

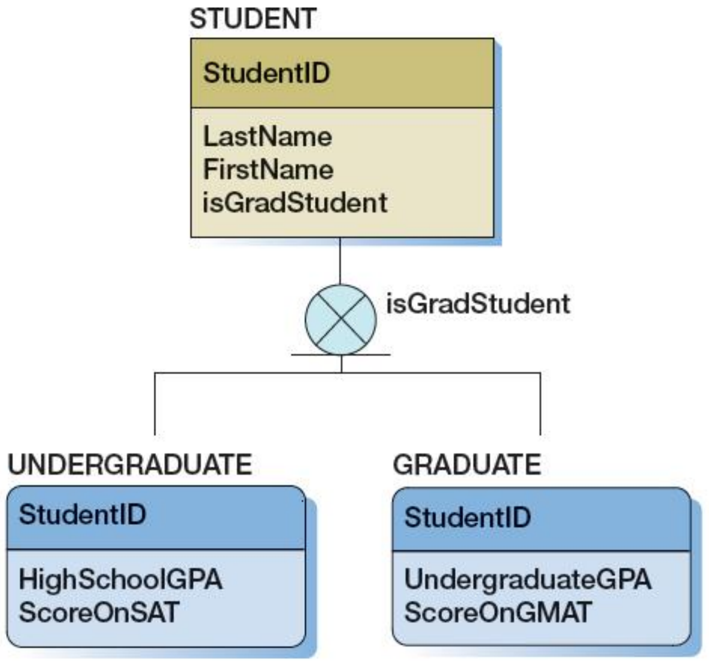


(a) Data Model with Multivalued Attributes from Figure 5-29

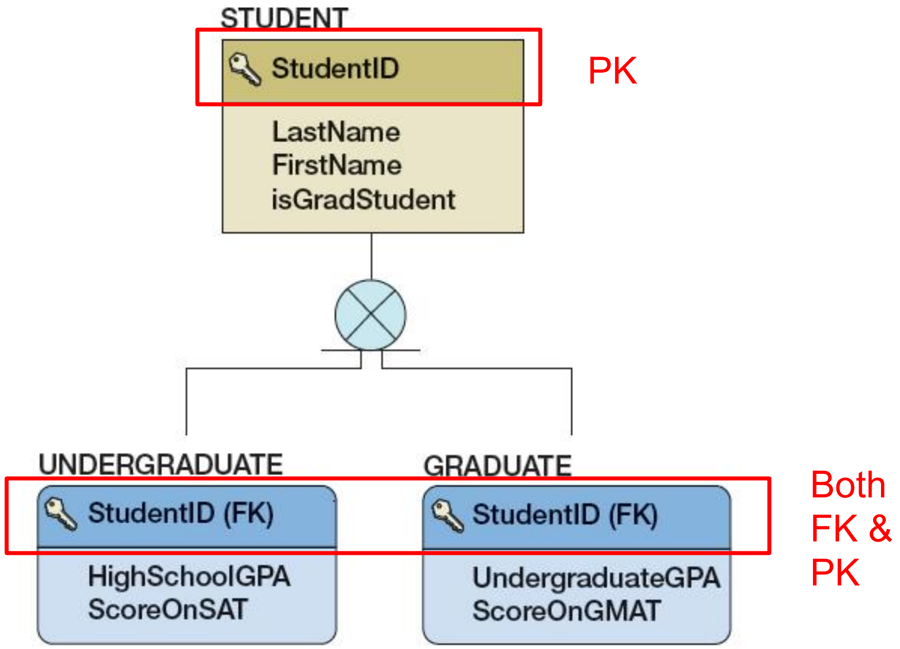


(b) Database Design to Store Multivalued Attributes

Subtype-Supertype



(a) Data Model of the Supertype/Subtype Relationship from Figure 5-20(a)



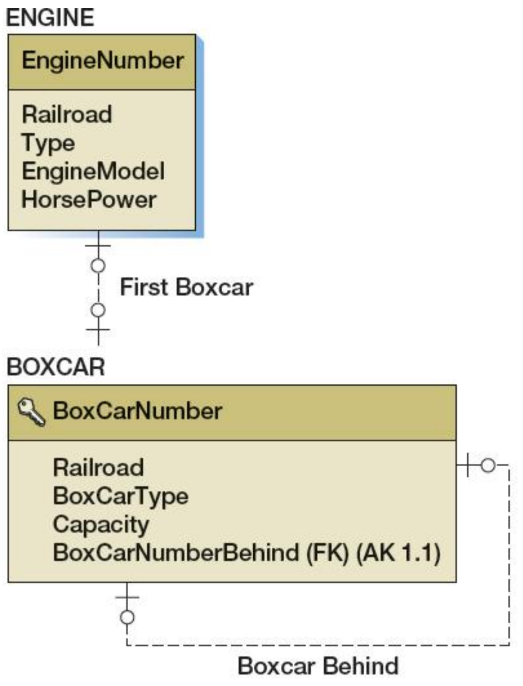
(b) Database Design for the Supertype/Subtype Relationship

Recursive Relationships

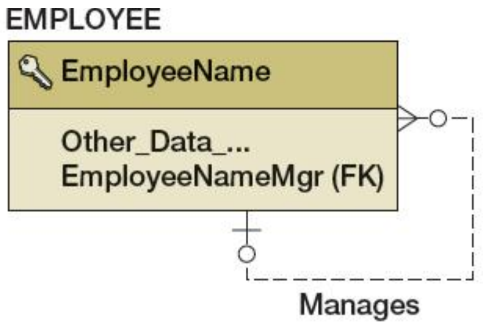
(Assuming 1:1 or 1:N relationship ...)

Just add an FK field for the relationship.

(N:M relationships are eliminated with associative entities in step 1.)



(b) Database Design for a 1:1 Recursive Relationship



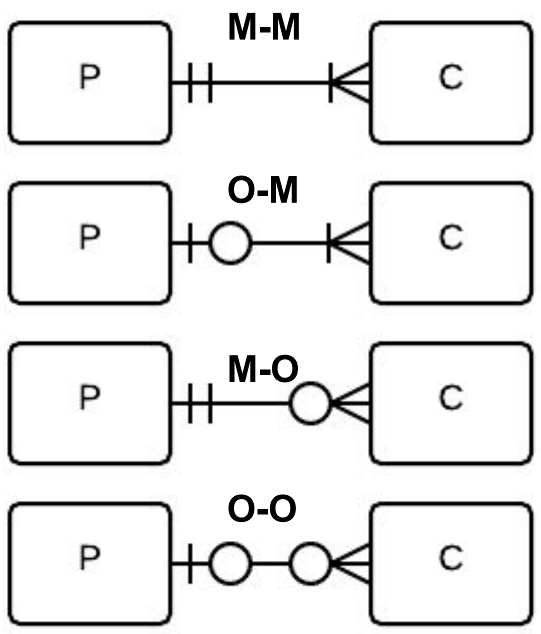
(b) Database Design for a 1:N Recursive Relationship

Optional/Mandatory Cardinalities

Pay attention to the minimum cardinality on either side of each relationship.

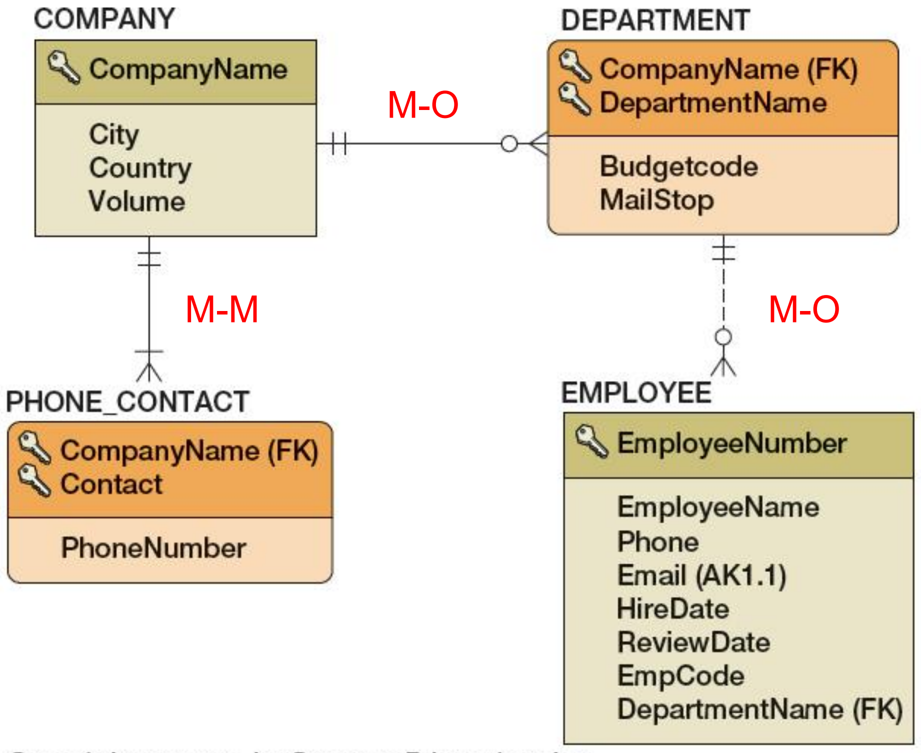
Note that **M**s here stand for **mandatory**, not **many.**

1:M is one to **many.**
***-M** means **mandatory** child



Nulls and Mandatory Pairs

- Optional FKs (**O-***) allow NULLs
- Mandatory FKs (**M-***) disallow NULLs.
- If both sides are mandatory (**M-M**) then the ***first*** entities must be created in pairs (like twins): one side cannot exist without the other, not even for a split second



Cascading Updates and Deletes

Cascading Updates: If a PK value changes, then any FKs that refer to it must also change to match.

- Surrogate key values never change, yet another reason to use them!

Cascading Deletes: When deleting a **parent entity** with one or more **child entities**, the children (often) must be deleted too.

- Happens when the relationship in the child-to-parent direction is mandatory (not optional).

Constraints Versus Triggers

Referential integrity constraints logically enforce mandatory relationships.

- If an **update-delete** action on the parent would cause a child to become parent-less, then either **prevent or cascade** it.
- Also, **prevent** the **insertion** of parent-less child entities.

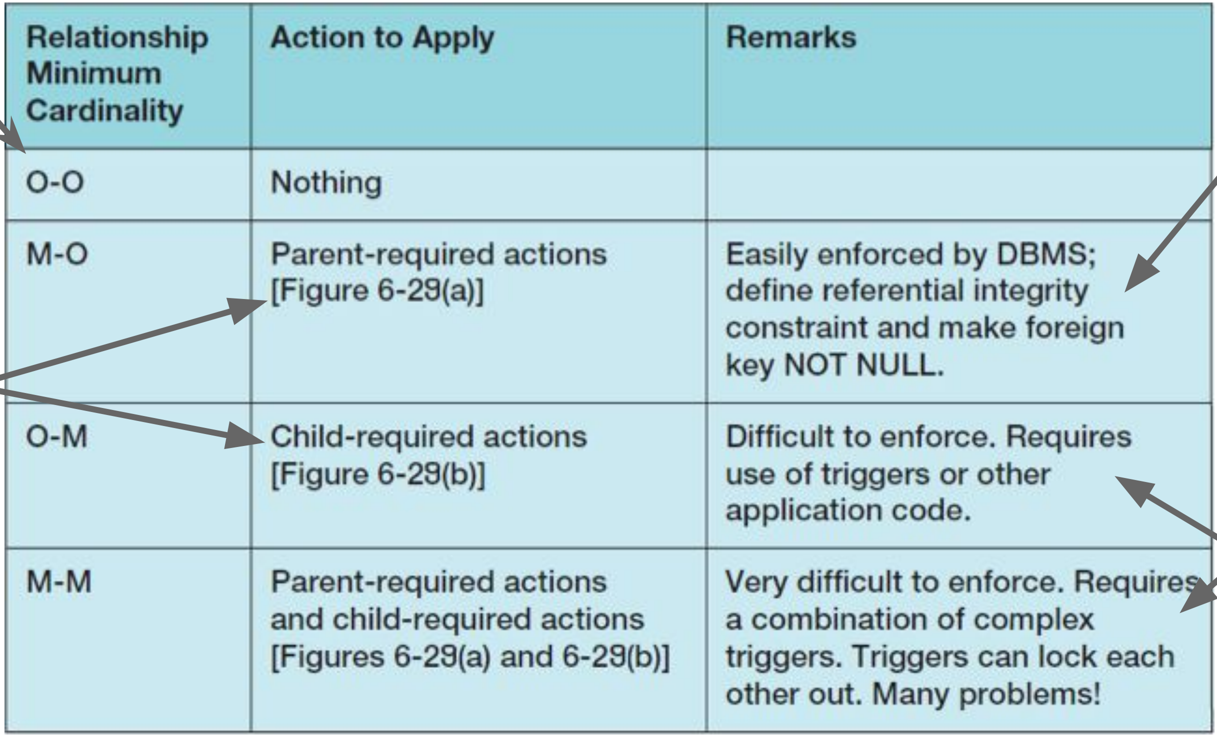
Triggers programmatically enforce mandatory relationships when referential integrity doesn't apply.

- If a **parent** must have **at least one child**, then a **trigger** would be used to prevent deletion of the last child.

Minimum Cardinality Rules

Parent-Child

Update or Delete



Relationship Minimum Cardinality	Action to Apply	Remarks
O-O	Nothing	
M-O	Parent-required actions [Figure 6-29(a)]	Easily enforced by DBMS; define referential integrity constraint and make foreign key NOT NULL.
O-M	Child-required actions [Figure 6-29(b)]	Difficult to enforce. Requires use of triggers or other application code.
M-M	Parent-required actions and child-required actions [Figures 6-29(a) and 6-29(b)]	Very difficult to enforce. Requires a combination of complex triggers. Triggers can lock each other out. Many problems!

Ref integrity constraints can force a cascade without extra code

No help from ref integrity. Need to enforce program-matically with triggers.

Table Designs to SQL DDL

Create, modify, and drop database tables

Data Definition Language (DDL)

DDL is used to **create, alter, or drop table** schemas (metadata).

Metadata changes cascade to the table data:

- Dropping a table schema deletes the data
- Modifying a table schema modifies the data

We use **Data Manipulation Language** (DML) to **create, update, or delete table data. We'll cover that next time.**

MySQL DDL Statements

Table DDL

- `CREATE / ALTER / DROP / TRUNCATE` **`TABLE`**
- `ADD / MODIFY / ALTER / DROP` **`COLUMN`**
- `ADD / DROP` **`PRIMARY KEY`**
- `ADD / DROP` **`FOREIGN KEY`**
- `CREATE / ADD / DROP` **`INDEX`**

Database DDL

- `CREATE / DROP DATABASE`

User DDL

- `GRANT / REVOKE`

Utility DDL

- `SHOW DATABASES`
- `USE`
- `SHOW TABLES`
- `SHOW COLUMNS`

Create and Drop Databases

- **Create =** Set up files for reading and writing
- **Drop =** Delete all traces of the database files
- Authentication maybe be configured after creation
- **Specific commands depend on the RBDMS**

`CREATE DATABASE` *database-name* ;

`DROP DATABASE` *database-name* ;

We can also drop tables, columns, keys, indexes, etc.

Show and Use Databases

```
SHOW DATABASES;
```

- Useful when you need to explore a MySQL Server instance from the command line.

`USE` *database-name*`;`

- Sets the current database; it's possible to use multiple databases but that's an advanced topic

Create Tables

Among the most complex SQL statements you will ever use

`CREATE TABLE` Statement

- Gives the table a name
- Declares all attributes (columns) and indexes

```
CREATE TABLE table-name (
   column-list
   PRIMARY KEY (pk-column-list)
);
```

CREATE TABLE Example

```
CREATE TABLE CREDITS (
  ID int(11) NOT NULL auto_increment,
  CCode varchar(1) default NULL,
  CName varchar(50) default NULL,
  MID int(11) default NULL,
  PRIMARY KEY  (ID)
);
```

These examples are MySQL dialect. SQLite uses slightly different data types but otherwise is similar.

Defining a Column (Attribute)

Syntax:

column-name data-type constraints,

Examples:

```
ID int(11) NOT NULL auto_increment,
MTitle varchar(255) default NULL,
Rating varchar(5) default NULL,
```

The trailing comma is omitted for the last column of the table.

SQL Data Types

- Data type selection is usually dictated by nature of data and by intended use
- Supported data types:
  - Number(L,D), Integer, Smallint, Decimal(L,D)
  - Char(L), Varchar(L), Varchar2(L)
  - Date, Time, Timestamp
  - Real, Double, Float
  - Interval day to hour
  - Many other types

SQL Column Constraints

- `NOT NULL`
  - Ensures that column does not accept nulls
- `UNIQUE`
  - Ensures that all values in column are unique
- `DEFAULT`
  - Assigns value to attribute when a new row is added
- `AUTO_INCREMENT`
  - Indicates the use of a serial number generator

Indices (and Keys)

- When primary key is declared, DBMS automatically creates a unique index
- Often need additional indexes
- Indexes can be created based on any selected attribute (don’t have to be unique)
- Composite index
  - Index based on two or more attributes
  - Often used to prevent data duplication

Declaring an Index / Key

- Usually declared during table creation (see below)
- Can also use the CREATE INDEX command

table name

```
CREATE TABLE MOVIES (
    ID int(11) NOT NULL AUTO_INCREMENT,
    MTitle varchar(255) DEFAULT NULL,
    Rating varchar(5) DEFAULT NULL,
    PRIMARY KEY (ID),
    INDEX (MTitle)
);
```

pk column

column name

Declaring a Foreign Key

- Declared after the keys and indices

table name

```
CREATE TABLE CREDITS (
    … other columns …
    MID int(11) DEFAULT NULL,
    PRIMARY KEY (ID),
    INDEX(MID),
    FOREIGN KEY (MID) REFERENCES MOVIES(ID)
);
```

table name

fk column

pk column

Cascading Deletes and Updates

```
CREATE TABLE CREDITS (
   … other columns …
    MID int(11) DEFAULT NULL,
    PRIMARY KEY (ID),
    INDEX(MID),
    FOREIGN KEY (MID) REFERENCES MOVIES(ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

Several possible response actions:

- **CASCADE** does as expected
- **RESTRICT** stops the update/delete if it causes a ref integrity error
- **SET NULL** sets the FK to null to avoid ref integrity problem

Altering Tables

Sometimes you can’t drop and start over

The ALTER TABLE statement

- **ALTER TABLE** commands can modify table structures after the tables are created
- Three basic options:
    - **ADD** adds a column
    - **MODIFY** changes column characteristics
    - **DROP** deletes a column
- Can also be used to:
    - Add table constraints
    - Remove table constraints

Adding a New Column

ALTER TABLE *table* **ADD COLUMN** *column-name column-def*;

table name

```
ALTER TABLE CREDITS ADD COLUMN
   AID INT(11);
```

column name

column definition

Modifying a Column

ALTER TABLE *table* **MODIFY COLUMN**
    column-name column-definition ;

```
ALTER TABLE CREDITS MODIFY COLUMN
  AID INT(11) NOT NULL;
```

Renaming a Column

ALTER TABLE *table* **CHANGE COLUMN**
old-name new-name column-definition;

```
ALTER TABLE MOVIES CHANGE COLUMN
    MRating Rating CHAR(5);
```

old column name

new column name

new column definition

Dropping a Column

ALTER TABLE table **DROP COLUMN** column-name;

```
ALTER TABLE CREDITS DROP COLUMN CName;
```

Adding an Index/Key

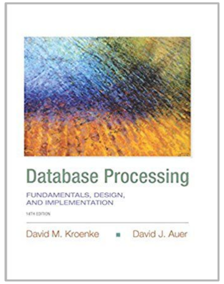
```
ALTER TABLE table ADD PRIMARY KEY (pk-columns);
ALTER TABLE table ADD INDEX (indexed-columns);
ALTER TABLE table
    ADD FOREIGN KEY (fk-column)
        REFERENCES ref-table (pk-column);

ALTER TABLE CREDITS
    ADD FOREIGN KEY (MID) REFERENCES MOVIES (ID);
```

MySQL Language Docs

- Every RDBMS implements it own dialect of the ANSI SQL standard
- The complete language reference manual for MySQL 5 is available at http://dev.mysql.com/doc/refman/5.7/en/sql-syntax.html
- If you have any question about how to do something, **RTFM for your DBMS**

Fairfield University DOLAN School of Business



Databases for Analytics

Kroenke / Auer
Chapters 6 and 7 (partial)