Data Modeling with the UML

Michael R. Blaha, DSc.
Modelsoft Consulting Corporation
E-mail: blaha@computer.com
Web: www.modelsoftcorp.com

Chicago DAMA
August 17, 2011

Section 1: Introduction

What is the UML?

The UML (Unified Modeling Language) is a graphical language for modeling software.

- The UML arose from the many object-oriented software development approaches of the 1990s.
- The purpose of the UML was to standardize object-oriented concepts and notation.
- The UML has been sponsored by the OMG (Object Management Group).

This talk will explain UML constructs through IDEF1X and examples.

Various UML Diagrams

The UML has a variety of diagrams including the following:

- **Class diagram**. Involves classes and relationships. Specifies data structure.
- **Object diagram**. Documents individual objects and links among objects. Shows examples of data structure.
- **Use case diagram**. Specifies high-level software functionality from the perspective of an end-user.
- **State diagram**. Concerns states and events that cause transitions between states. Describes the discrete, temporal behavior of objects.
- **Activity diagram**. Shows the workflow for an individual piece of functionality.
- **Sequence diagram**. Shows how processes interact, with whom and what, and in what order.

The remainder of this lecture covers the class model, the model that is most relevant for databases.

Why Use the UML? What Is It Good For?

The UML class model is suitable for conceptual data models.

- Communicating with business experts.
  - The UML avoids confusing database details.
    - Can show attributes without addressing keys.
    - Avoid distinction between independent and dependent entities.
    - Can show role names without showing attributes.
    - Can show selected relationships by package / subject area.
- Communicating with programmers.
  - Most programmers are familiar with the UML and like it.
- Abstraction.
  - It is helpful to suppress database details when thinking about deep concepts such as data modeling patterns.
- Behavior.
  - UML operations can summarize stored procedures and SOA services.

What Are Weaknesses of the UML?

The UML class model does not address database details.

- Does not address database design.
  - The UML notation does not cover database design.
  - Most UML tools also lack database design support.
    - Enterprise Architect has some database support. (Need to investigate further.)
- Overemphasizes programming jargon.
  - The UML creators focused on programming and ignored databases.
  - Ironically the programming jargon is superficial and the UML has much to offer for database applications.
- Has little use by database practitioners.
  - Many database practitioners know about the UML.
  - There is a chasm between programming and database cultures.

My Development Process

- Construct a model of conceptual intent (using the UML).
  - Use a UML tool such as Enterprise Architect.
  - Build a UML class model in conjunction with business experts.
  - Write an explanation of the business intent.
- Construct a database design model (using IDEF1X or another database notation).
  - Use a database design tool such as ERwin or Dezign.
  - Often must manually re-key the model.
  - Observe the organization's database design conventions (naming, foreign key enforcement, triggers, etc.)
  - Generate SQL code.
- Maintain both models and explanation as revisions occur.
- Use agile development.

Any Comments?

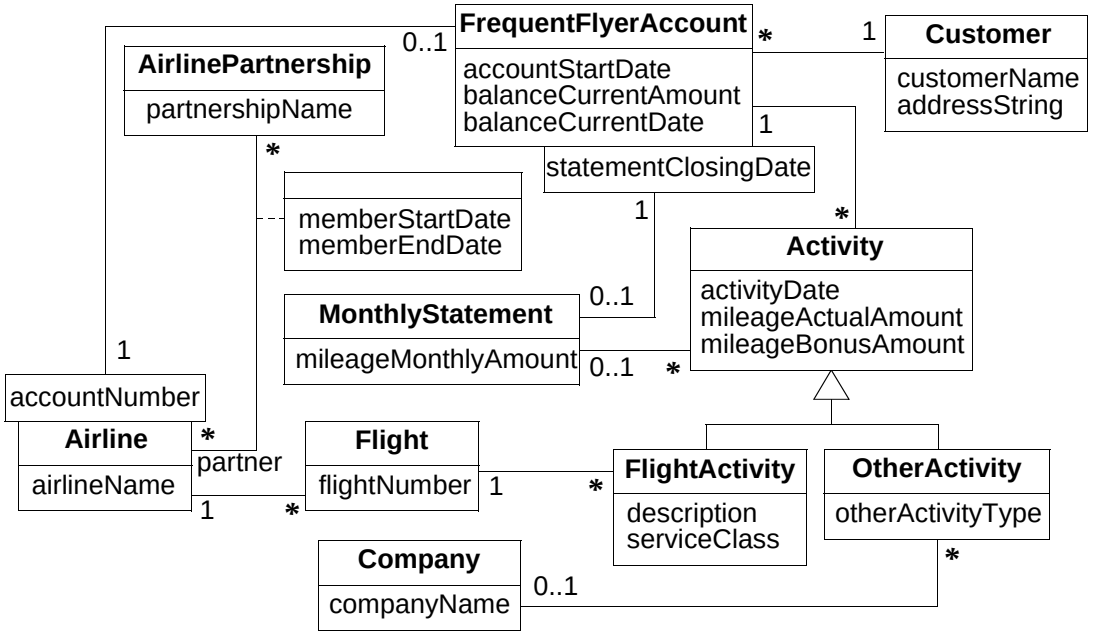
- Does anyone here use the UML?
  - If so, what UML tools?
- Have I missed any major UML strengths and weaknesses?
- Does anyone construct data models interactively with business customers?

Note that this seminar focuses on transactional (OLTP) applications.

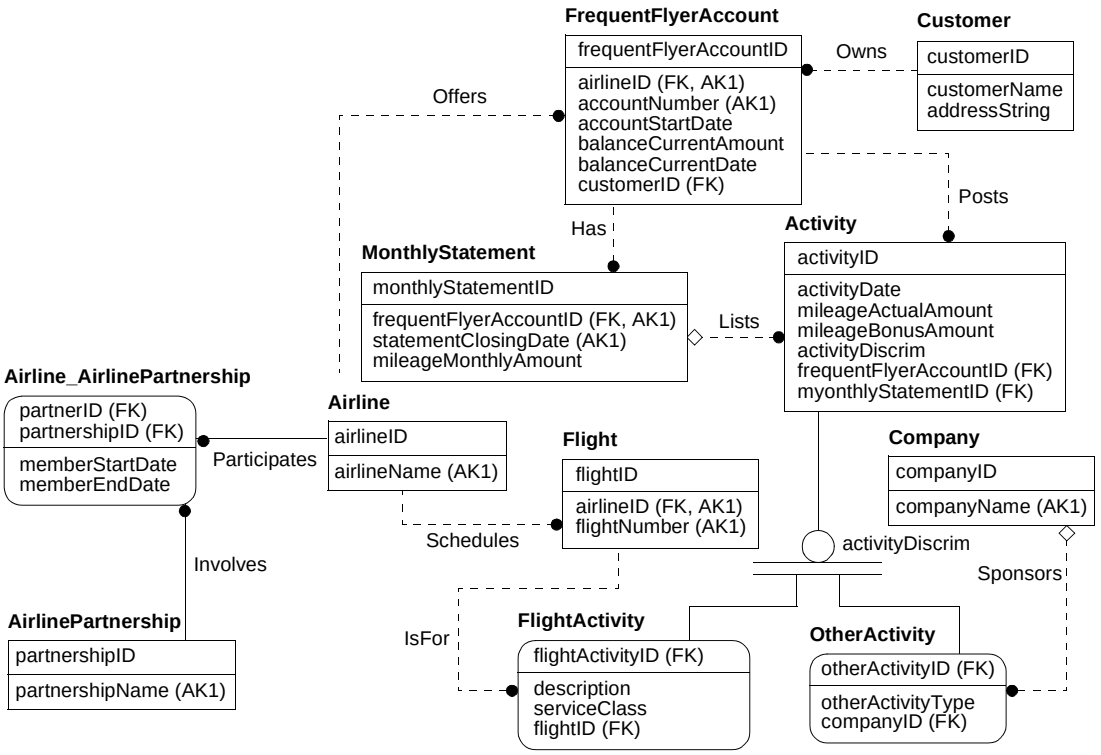
There is no compelling reason to use the UML for data warehouse (OLAP) applications.

Section 2: A Sample Data Model

A UML Data Model

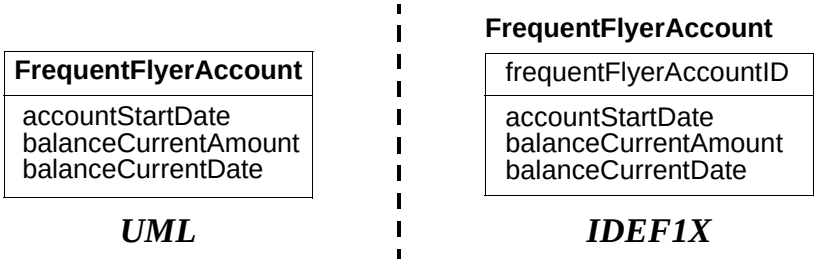


An IDEF1X Data Model



Section 3: UML Class

UML Class



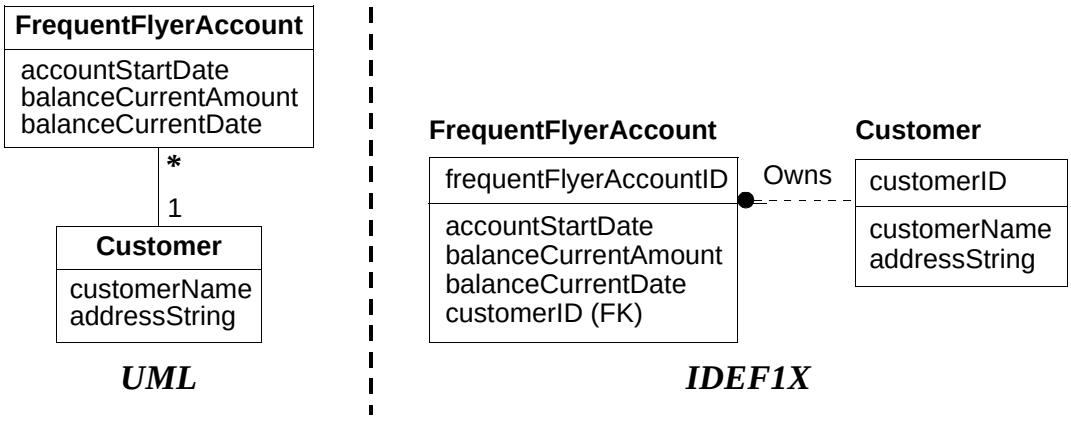
- ***Object*** — a concept, abstraction, or thing with identity that has meaning for an application.
- ***Class*** — a description of a group of objects with the same attributes, operations, kinds of relationships, and semantic intent.
  - **UML notation**. A box with the name of the class in the top portion.
  - **Examples**. *Airline*, *FrequentFlyerAccount*, and *Activity*. (Section 6 discusses identity.)

UML Attribute

- ***Value*** — a piece of data. Values have no identity.
- ***Attribute*** — a named property of a class that describes a value held by each object of the class. An attribute is a "slot" for data.
  - Analogy — class::object as attribute::value.
  - **UML notation**. Listed in a second portion beneath the class name.
  - **Example**. *FrequentFlyerAccount* has three attributes.
- ***Operation*** — a function or procedure that may be applied to or by objects in a class.
  - **UML notation**. Listed in the third portion of class box. (Not shown.)
  - **Example**. A stored procedure could update *balanceCurrentAmount* each time an *Activity* posts.

Section 4: UML Association

UML Association



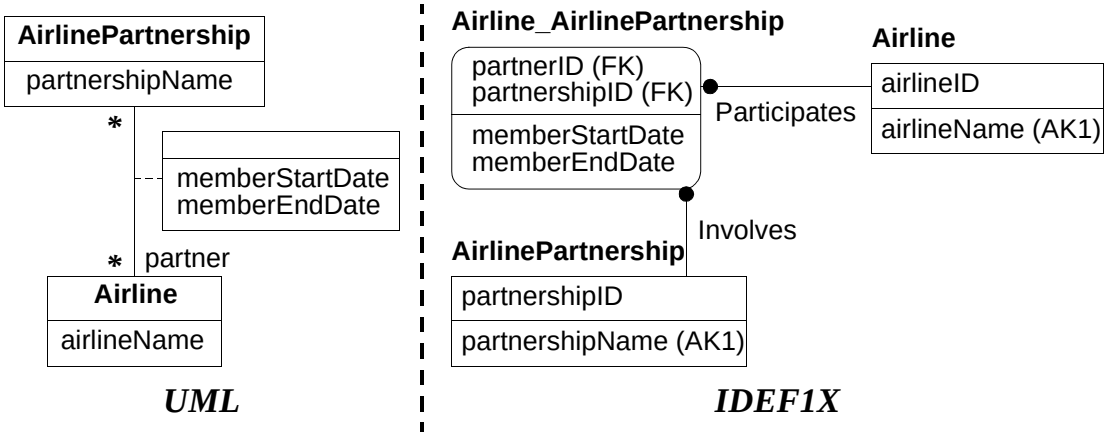
UML Association (continued)

- ***Link*** — a physical or conceptual connection among objects.
- ***Association*** — a description of a group of links with common structure and semantics.
  - The links of an association connect objects from the same classes.
  - An association describes a set of potential links in the same way that a class describes a set of potential objects.
  - **UML notation**. A line (possibly with multiple line segments) connects the related classes.
  - **Example**. The line between *FrequentFlyerAccount* and *Customer*.

UML Association End

- A binary association has two ends.
  - Each end can have a name and multiplicity.
- ***End name*** — an alias for a class in an association.
  - **UML notation**. A legend next to the class–association intersection.
  - **Example**. An *Airline* is a *partner* in an *AirlinePartnership*.
- ***Multiplicity*** — the number of instances of one class that may relate to a single instance of an associated class.
  - Multiplicity constrains the number of related objects.
  - Often called “cardinality” (though mathematically incorrect).
  - **UML notation**. Usually “1”, “0..1”, and “*” (“many” — zero or more).
  - **Example**. A *FrequentFlyerAccount* has one *Customer*.
    A *Customer* can have many *FrequentFlyerAccounts*.

UML Association Class



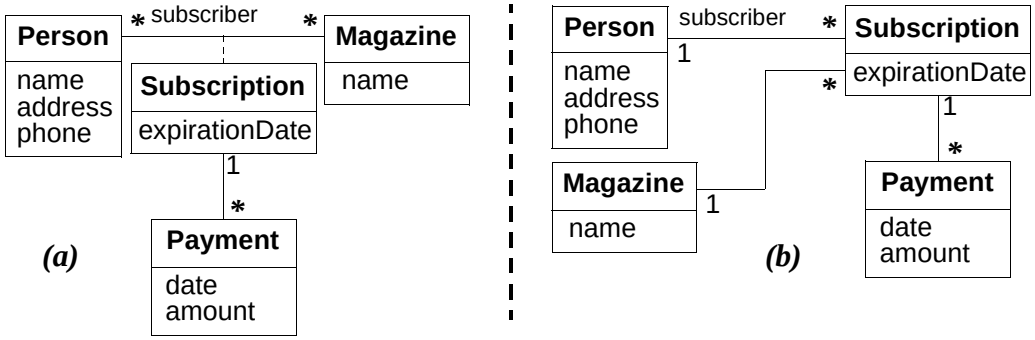
- ***Association class*** — an association that is also a class.
  - Like an association, the occurrences of an association class derive identity from objects of the constituent classes.
  - Like a class, an association class can have attributes and operations and participate in associations.

UML Association Class (continued)

- **UML notation**. A box connected to the association with a dotted line.
- **Example**. The association class between *AirlinePartnership* and *Airline* has two attributes.

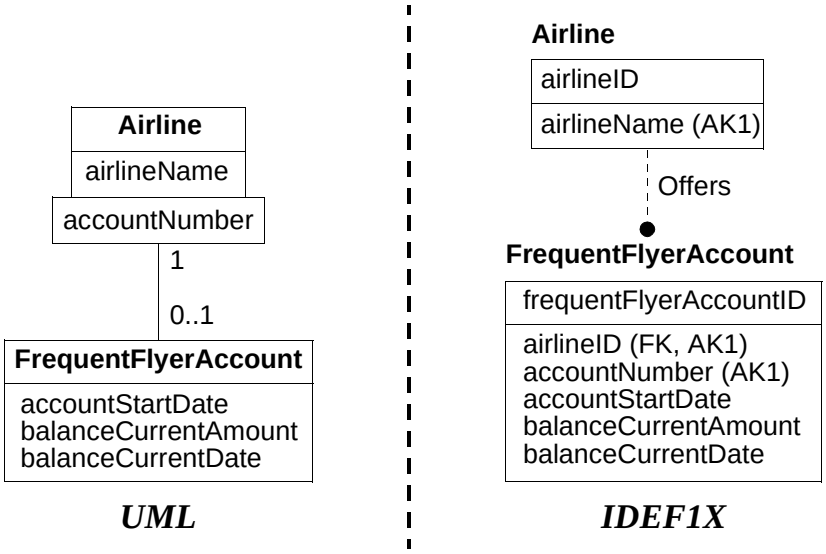
Problem 1 — Association Class

Compare the following models. Which one is better? The left model represents *Subscription* as an association class; the right model treats *Subscription* as an ordinary class.



A person may have multiple magazine subscriptions. A magazine has multiple subscribers. For each subscription, it is important to track the date and amount of each payment as well as the current expiration date.

UML Qualified Association



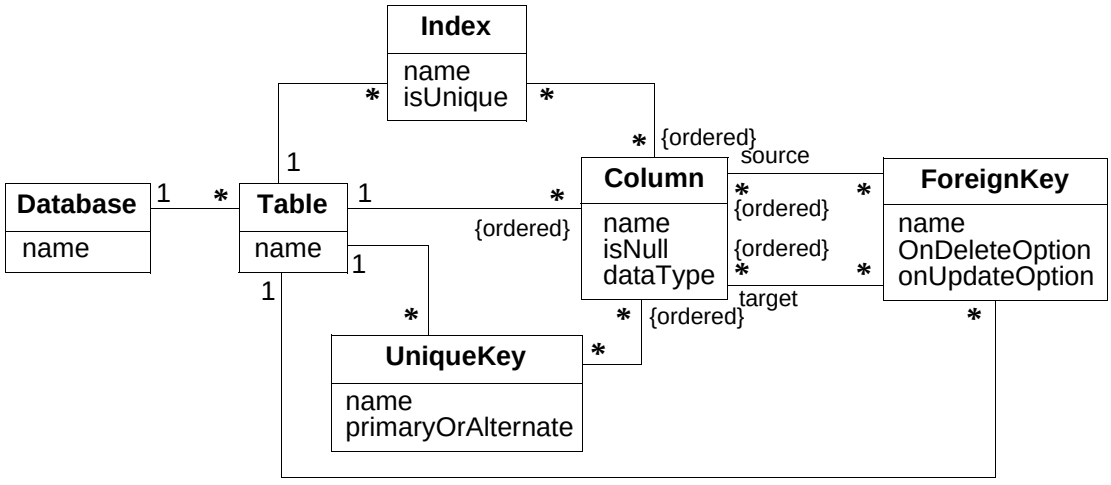
- ***Qualified association*** — an association in which an attribute called the ***qualifier*** partially or fully disambiguates the objects for a "many" association end.

UML Qualified Association (continued)

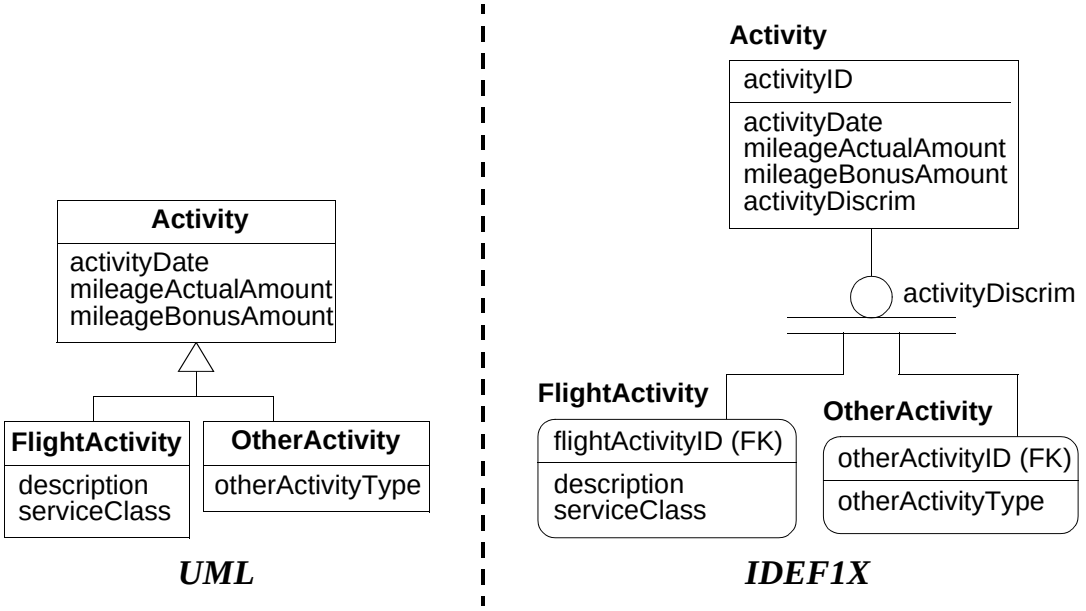
- The qualifier selects among the target objects, reducing the effective multiplicity, often from “many” to “one”.
- Names are often qualifiers.
- **UML notation**. A small box on the association line by the source class. The source class plus the qualifier yields the target class.
- **Example**. The *accountNumber* for a *FrequentFlyerAccount* is unique within the context of the issuing *Airline*.
- **Example**. The *statementClosingDate* for a *MonthlyStatement* is unique within the context of a *FrequentFlyerAccount*.

Problem 2 — Qualified Association

Restate the following model to use qualifiers.



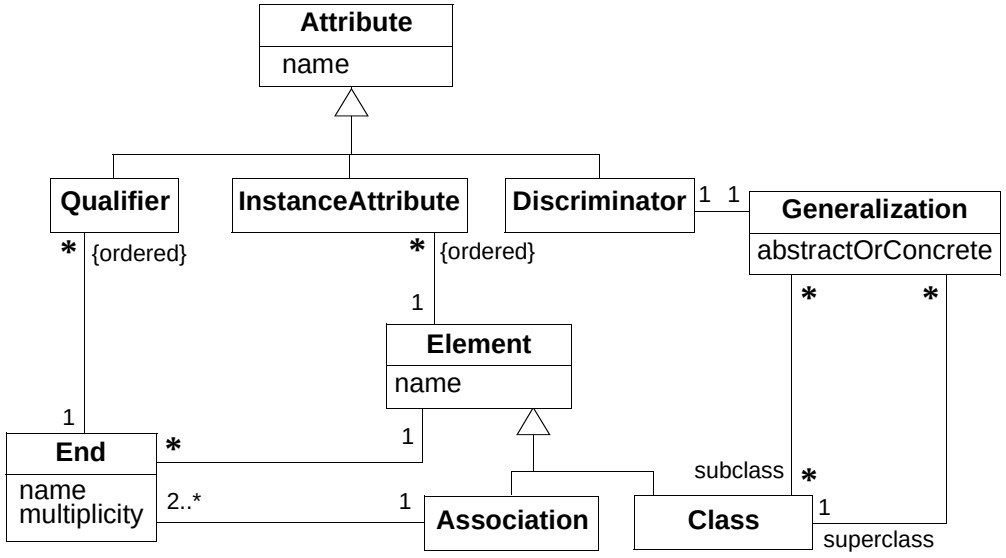
UML Generalization



UML Generalization (continued)

- ***Generalization*** — the relationship between a class (the ***superclass***) and one or more variations of the class (the ***subclasses***).
  - Generalization structures the description of objects.
  - The superclass has general data.
  - The subclasses have specific data.
  - **UML notation**. A large hollow arrowhead points to the superclass. Lines fan out towards the subclasses.
  - **Example**. An *Activity* can be a *FlightActivity* or an *OtherActivity*.

Abridged Class Metamodel



Section 6: Identity

Identity — the property that distinguishes an entity from all others.

- ***Existence-based identity*** — a system-generated identifier.
  - Each object primary key is a single field, small, and uniform in size.
  - The database has a consistent approach to identity.
  - However... The database can be more complex to inspect and debug.
- ***Value-based identity*** — a unique combination of real-world attributes.
  - Primary keys have intrinsic meaning.
  - However... A primary key change must propagate to foreign keys.
  - However... Some objects lack value-based identifiers.
  - However... Multi-field primary keys can be unwieldy.
- Mix of existence-based and value-based.
  - However... An irregular approach.

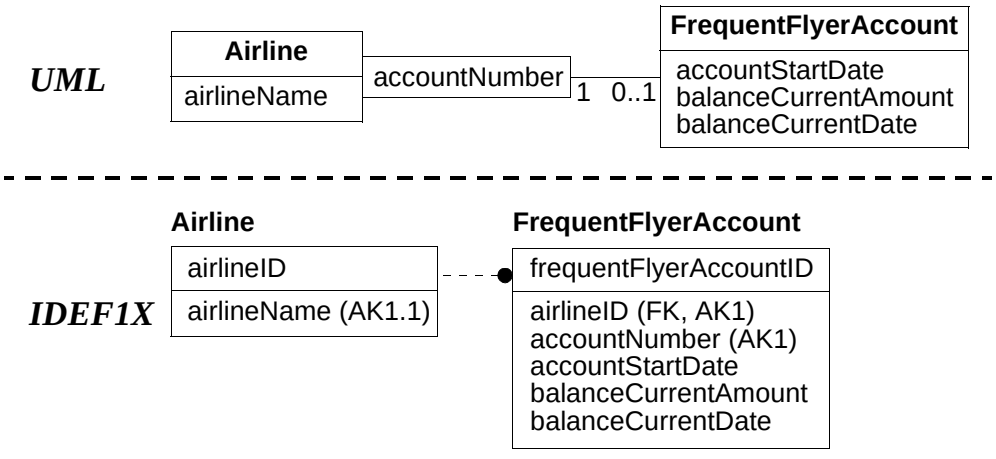
Approaches to Identity (continued)

- ***Surrogate identity*** — identification via related entities.
  - **Example**. Identifying a person via a passport, driver's license, or identity card.
  - However... Violates normal forms.

I normally use existence-based identity. (As a minor variation, it is OK to use value-based acronyms for enumeration tables.)

UML Qualifier

- ***UML qualifier*** — an attribute that distinguishes among "many" objects.
  - **Example**. An *Airline* has many *FrequentFlyerAccounts*. An *Airline* plus an *accountNumber* yield at most one *FrequentFlyerAccount*.



Candidate Key (Alternate Key)

- ***Candidate key*** — a combination of one or more fields that uniquely identify the records in a table.
  - The set of fields in a candidate key must be minimal.
  - No field in a candidate key can be null.
  - The DBMS can guarantee the uniqueness of candidate keys.
  - **UML notation**. None...
  - **Examples**. *AirlineName* is a candidate key.

Of course, a primary key is an arbitrary choice of candidate key.

Name

Names are prominent in models. A name is a word or phrase that designates a person or thing.

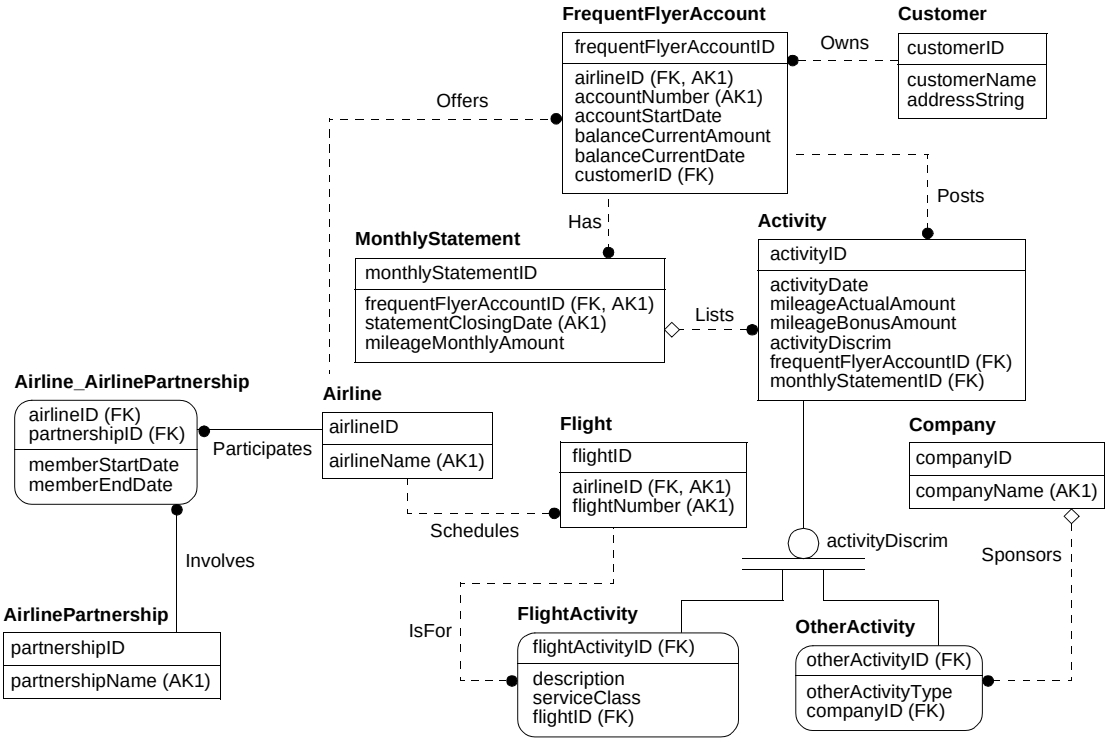
There are four scenarios for how names can be used.

- **Unique names**. Some names are unique and resolve to a single entity. *Airline name* is globally unique.
- **Unique names within a context**. Other names are not unique on their own but are unique when combined with a parent entity (UML qualifiers). The *accountNumber* provides the unique name for a *FrequentFlyerAccount* within the context of an *Airline*.
- **Non-unique names**. Still other names provide important description but alone cannot find an entity. For example, person names are important, but insufficient for finding an individual person.
- **Multiple unique names**. Some entities have multiple names. For example, propylene is known as *propylene* and C_3H_6.

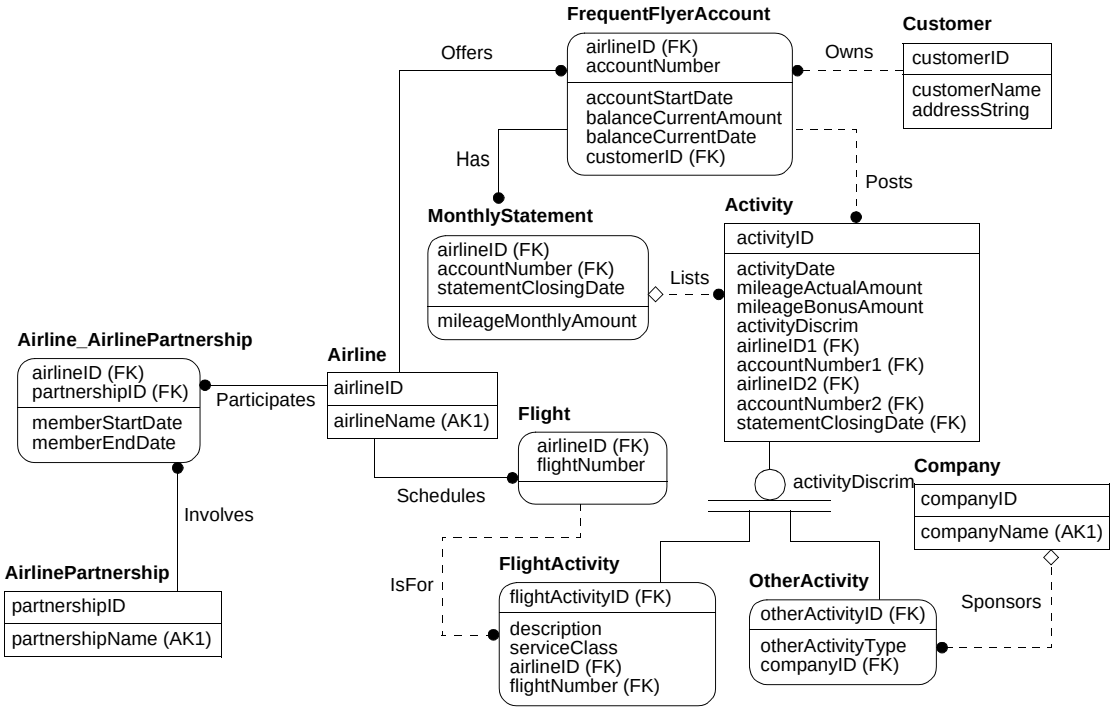
Structured Field

- ***Structured field*** — a field that is composed from constituent pieces with a specified grammar.
  - Structured fields are synthetic but the pieces have meaning.
  - Many structured fields have standard protocols.
  - **Examples**. Mechanical parts and items for commerce often have structured fields that provide identity.
  - **Example**. The UPC (Universal Product Code) has twelve digits.
    - The first digit indicates the kind of item—general merchandise, random-weight item, health item, in-house item, and coupon.
    - The next five digits denote the vendor.
    - Digits seven through eleven identify an item within the context of a vendor.
    - The last digit is a check digit.

IDEF1X Notation — An Example



IDEF1X Notation — Another Design



Section 7: Database Design

Database Design Topics

- Indexes
- Referential integrity
- Enumerations
- Views

Index

Indexes provide the primary means for tuning a relational database.

- Normally you should define a unique index for each primary and candidate key.
  - Most relational DBMSs create unique indexes as a side effect of SQL primary key and unique constraints.
- You should also build an index for each foreign key that is not subsumed by a primary key or candidate key constraint. Foreign keys implement associations and generalizations.
  - The indexes on foreign keys and primary keys enable fast traversal of a class model.
- You should deliver your initial schema with indexes and referential integrity. You will frustrate users if you deliver a slow implementation.
  - It is straightforward to incorporate them into your initial schema.
- The DBA may define additional indexes to further speed frequently asked queries.

Referential Integrity

- Many relational DBMSs now support referential integrity. Use it!
- Define referential integrity actions for deletes. You can partially infer them from the class model.
- Given existence-based identity, you do not need referential integrity actions for updates.
- Don’t use triggers to implement referential integrity.

Implementing Referential Integrity for Deletion

- Generalization.
  - Normally cascade the effect of a deletion.
  - The RDBMS can propagate deletion downwards from the superclass to the subclasses. However, a RDBMS cannot propagate deletion upwards from the subclass towards the superclass.
- Buried association, minimum multiplicity of zero.
  - Normally set the foreign key to null.
  - Sometimes forbid deletion.
- Buried association, minimum multiplicity of one.
  - Cascade the effect of a deletion.
  - Or forbid deletion.
- Association table.
  - Normally cascade deletions to the records in an association table.
  - Sometimes forbid a deletion.

Enumeration

- An ***enumeration*** is a data type that has a finite set of values.
  - *ServiceClass* is first, business, or coach.
- You should carefully document enumerations.
- Do not use a generalization to capture the values of an enumerated attribute.
  - Use generalization only when at least one subclass has attributes, operations, or associations that do not apply to the superclass.
- Options for implementing enumerations.
  - One or more dedicated enumeration tables.
    Involves some work defining mechanisms, but a good approach.
  - Check constraints.
    A good approach for constraints that seldom change.
  - Enforcing enumerations with programming code.
    A poor option that many developers use anyway.

View

- A ***view*** is a database table that is computed as needed from a SQL query.
- Views can be used to consolidate the generalization levels that describe different aspects of an object.
- Many database engines support updatable views when they are theoretically possible. Often you can not only read through views, but also write through views.

Section 8: Conclusion

Agile UML Data Modeling

- See my YouTube videos for an example of data modeling using the UML.
  - Develop software for tracking library loan records.
  - http://www.youtube.com/view_play_list?p=EE77921A75E846EB
  - I routinely perform agile UML data modeling. I do live, interactive sessions with business customers and it works well.

Seeking Your Advice...

I may write a book explaining the UML class model in terms of IDEF1X — along the lines of this talk.

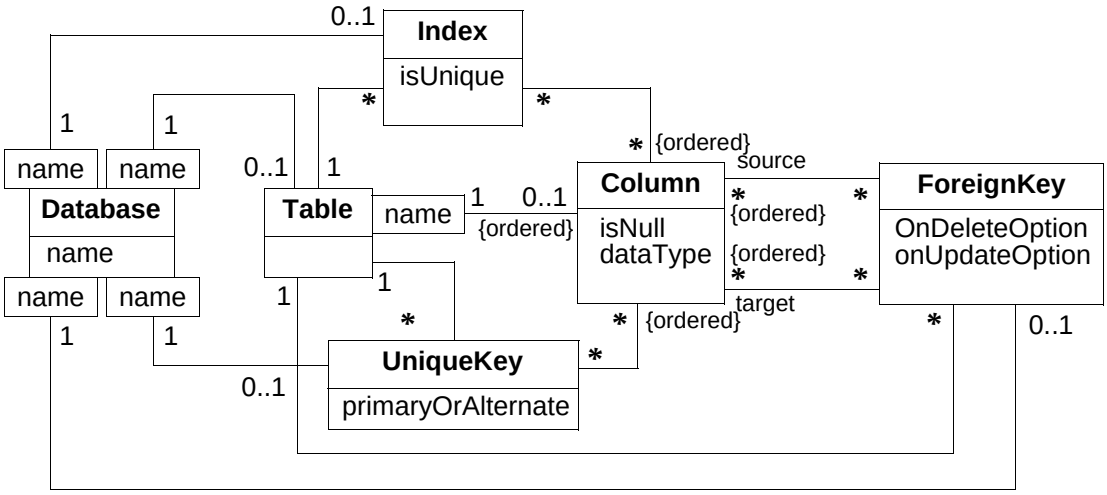
- Good idea?
- Bad idea?
- Suggestions?

Answer 1

- Model (a) states that a subscription has derived identity. Model (b) gives subscriptions more prominence and promotes subscription to a class.
- The (b) model is better.
  - Most copies of magazines have subscription codes on their mailing labels; this could be stored as an attribute.
  - The subscription code is intended to identify subscriptions; subscriptions are not identified by the combination of a person and a magazine so we should promote *Subscription* to a class.
  - Furthermore a person might have multiple subscriptions to a magazine; only the (b) model can readily accommodate this.

Answer 2 — Qualified Association

Restate the following model to use qualifiers.



Without qualifiers, the scope of uniqueness is not obvious.