NUS | Computing
National University of Singapore

CS2102: Database Systems

Lecture 4 — Entity Relationship Model (ER Model)

Errata

- Notation for renaming in RA expressions
  - Lecture

    $$\rho_{\text{new} \leftarrow \text{old}}$$

  - Tutorial 2 (solutions)

    $$\rho_{\text{old} \leftarrow \text{new}}$$

Quick Recap: SQL for Creating Databases

- Data Definition Language (DDL)
  - Create, modify and drop tables
    to implement a given DB schema
  - Specify integrity constraints
    (e.g., **NOT NULL**, **PRIMARY KEY**, **FOREIGN KEY**, **CHECK**)

- Data Manipulation Language (DML)
  - Insert, update and delete data from tables

Employees (<u>id: **integer**</u>, name: **text**, age: **integer,** role**: text**)

```
CREATE TABLE Employees (
        id      INTEGER PRIMARY KEY,
        name    VARCHAR(50) NOT NULL,
        age     INTEGER,
        role    VARCHAR(50)
);
```

Employees

id	name	age	role

```
INSERT INTO Employees VALUES
        (101, 'Sarah', 25, 'dev')
        (102, 'Judy', 35, 'sales');
```

Employees

id	name	age	role
101	Sarah	25	dev
102	Judy	35	sales

We Sneakily Skipped a Step

- Open questions:
  - Where does the database schema come from?
  - What tables with which attributes do we need?
  - What data integrity constraints are required?
  - Table names, attribute names, data types, …?

➜ Database Design Process

Solution

- Storing the "dob" instead of "age" is arguably the preferred approach
- The value of "age" changes each year (no really a big deal)
- "dob" provides more detailed information compared the "age"

Which table is "better"?

```
CREATE TABLE Employees (
        id      INTEGER PRIMARY KEY,
        name    VARCHAR(50) NOT NULL,
        age     INTEGER,
        role    VARCHAR(50)
);
```

or

```
CREATE TABLE Employees (
        id      INTEGER PRIMARY KEY,
        name    VARCHAR(50) NOT NULL,
        dob     DATE,
        role    VARCHAR(100),
        phone   INTEGER
);
```

Database Design Process — 6 Common Steps

Requirement Analysis

- Identification and collection of user needs
- e.g., data /application / performance requirements

Conceptual DB Design

- Capturing requirements using a conceptual model
- RDBMS: **Entity Relationship Model (ER Model)**

Logical DB Design

- Mapping conceptual model to logical schema of DBMS
- RDBMS: Entity Relationship Model → Relational Schema

Schema Refinement

- Checking schema / tables for redundancies and anomalies

Physical DB Design

- Implementing database based on final data schema
- Consideration of performance requirements

Security Design

- Identification users and user groups and their permissions to access which parts of the data

Overview

- **Entity Relationship Model**
  - Overview + ER diagrams
  - Entity sets and attributes
  - Relationship sets
  - Cardinality & participation constraints

- Relational Mapping
  - From ER diagram to database tables

- Extended notations for ER diagrams
  - ISA hierarchies: generalization/specialization
  - Aggregation

Requirement Analysis: Online Airline Reservation System (OARS)

Users need to be able to make bookings from an origin to a destination airport which may comprise multiple connecting flights. Each flight has a flight number, the origin and destination airport, the distance in kilometers, the departure and arrival time, and the days of the week the flight is in operation.

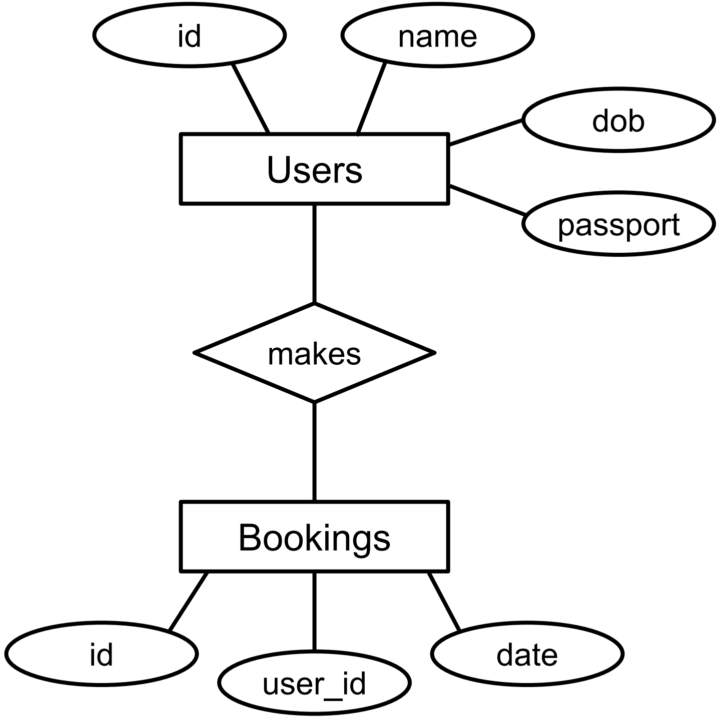
A flight instance is the actual scheduled flight on a given day together with the assigned aircraft type. For example, flight SQ231 flies daily from Singapore to Sydney, typically with a Boeing 777-300ER (code: B77W).

For a valid booking, we need the user's name, sex, address, phone number(s), and the passport number. Users are only able to pay via credit card. When making a booking, the user can select the class, the seat number, as well as meal preferences (if available).

Entity Relationship Model

- ER Model
  - Most common model for conceptual database design
  - Developed by Peter Chen (1976)
  - Visualized using **ER diagrams**

- Core concepts
  - All data is described in terms of **entities** and their **relationships**
  - Information about entities & relationships are described using **attributes**
  - Certain data constraints can be described using additional annotations

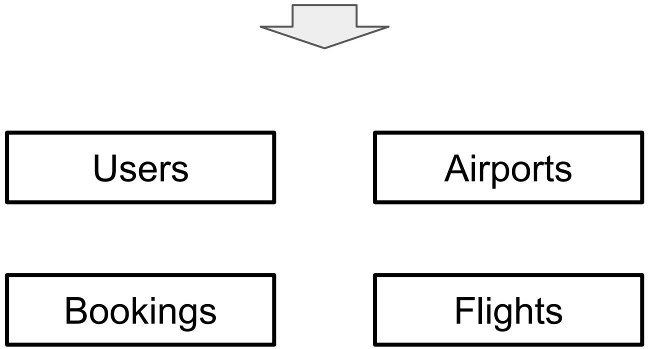


Entities and Entity Sets

- **Entity**
  - Real-world things or objects that are distinguishable from other objects
    (e.g., an individual user, airport, flight, or booking)

- **Entity Set**
  - Collection of entities of the same type
  - Represented by rectangles in ER diagrams
  - Names are typically nouns

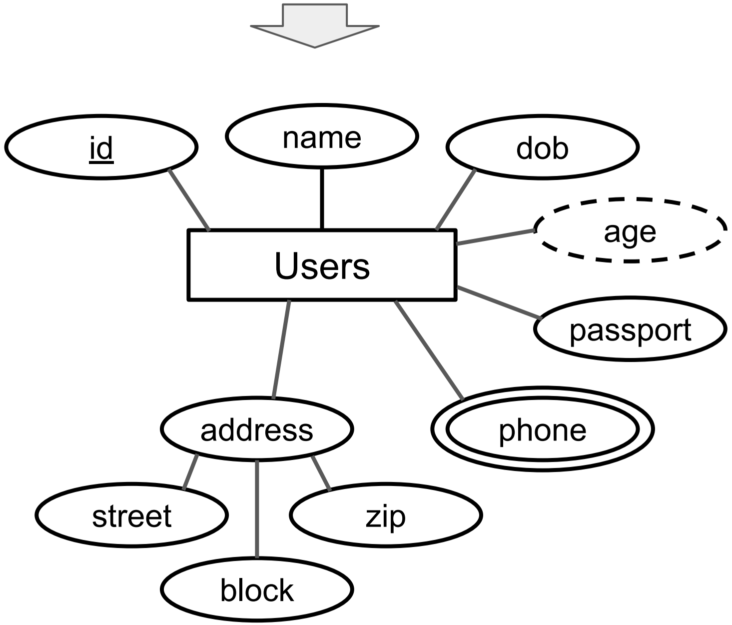
__Users__ need to be able to make __bookings__ from an origin to a destination __airport__ which may comprise multiple connecting __flights__. Each flight has a flight number, [...]



Attributes

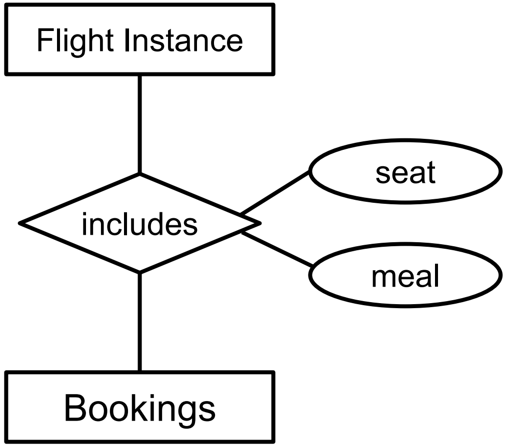
- **Attribute**:
  - specific information describing an entity
  - represented by an oval in ER diagrams
- 4 subtypes of attributes
  - **Key attribute(s)**: uniquely identifies each entity (oval with the attribute name(s) underlines)
  - **Composite attribute**: composed of multiple other attributes (oval comprising of ovals)
  - **Multivalued attribute**: may consisting more than one value for a given entity (double-lined oval)
  - **Derived attribute**: derived from other attributes (dashed oval)

For a valid booking, we need the ***user's name***, ***sex***, ***address***, ***phone number(s)***, *and the* ***passport number***.
Users are only able to pay via credit card. *[...]*



Relationships and Relationship Sets

- **Relationship**
  - Association among two or more entities
- **Relationship Set**
  - Collection of relationships of the same type
  - Represented by diamonds in ER diagrams
  - Can have their own attributes that further describe the relationship
  - Names are typically verbs

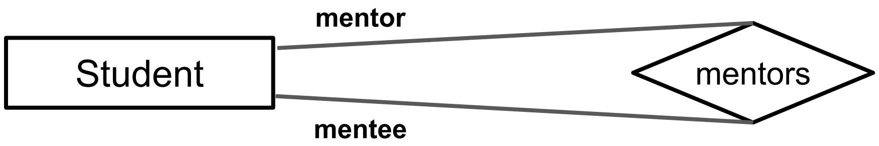


- Additional annotations to further specify relationships
  - Roles, degree, cardinalities, participation, dependencies

Relationship Roles

- **Role**
  - Descriptor of an entity set's participation in a relationship
  - Most of the time implicitly given by the name of the entity sets
  - Explicit role label only common in case of ambiguities
    (typically in case the same entity sets participates in the same relationship more than once)

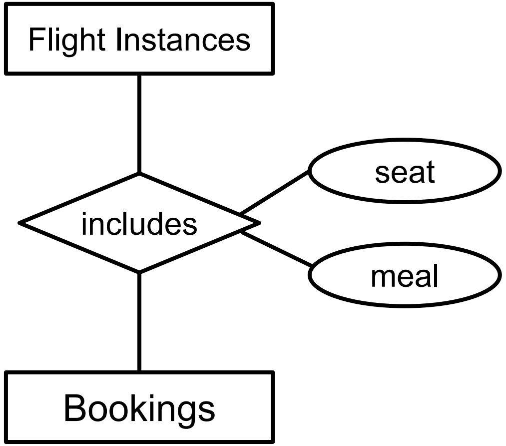
- Example: Students can mentor other students



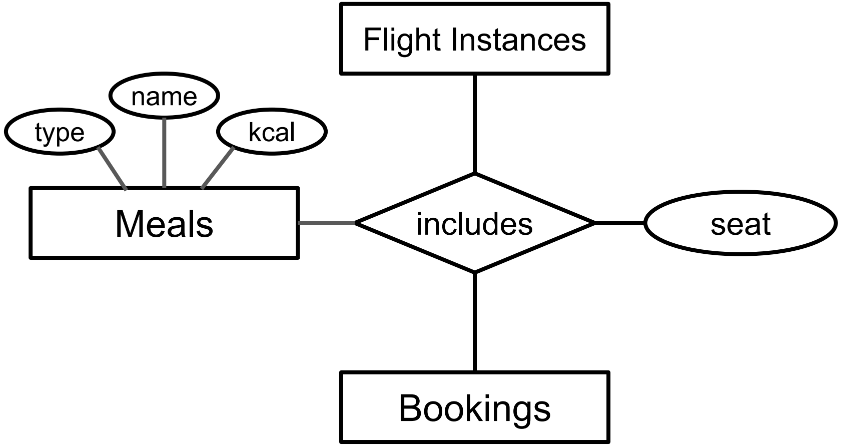
Degree of Relationship Sets

- **Degree**
  - In principle, no limitation how many entity roles participate in a relationship
  - An *n*-ary relationship set involves *n* entity roles ➜ *n* = degree of relationship set

n = 2 ➜ binary relationship set

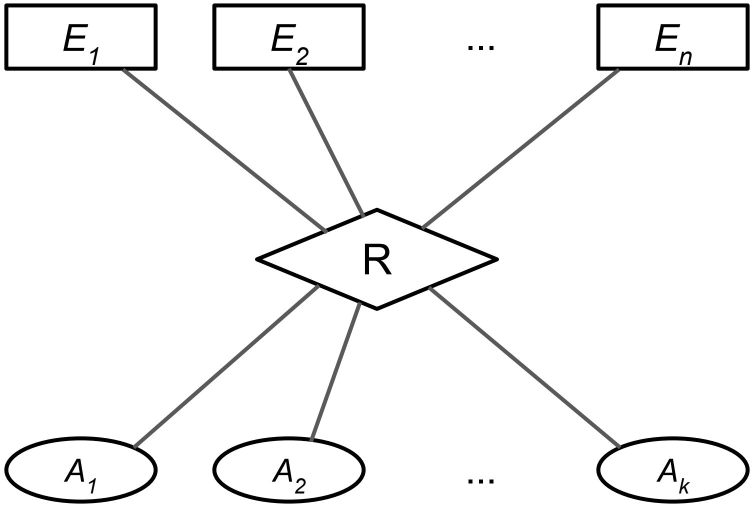


n = 3 ➜ ternary relationship set



Degree of Relationship Sets

- General n-ary relationship set R
  - *n* participating entity sets $E_1, E_2, \ldots, E_n$
  - *k* relationship attributes $A_1, A_2, \ldots, A_k$



"In typical modeling, binary relationships are the most common and relationships with n>3 are very rare" - Peter Chen (2009)

Overview

- **Entity Relationship Model**
  - Overview + ER diagrams
  - Entity sets and attributes
  - Relationship sets
  - **Cardinality & participation constraints**
- Relational Mapping
  - From ER diagram to database tables
- Extended notations for ER diagrams
  - ISA hierarchies: generalization/specialization
  - Aggregation

Cardinality Constraints

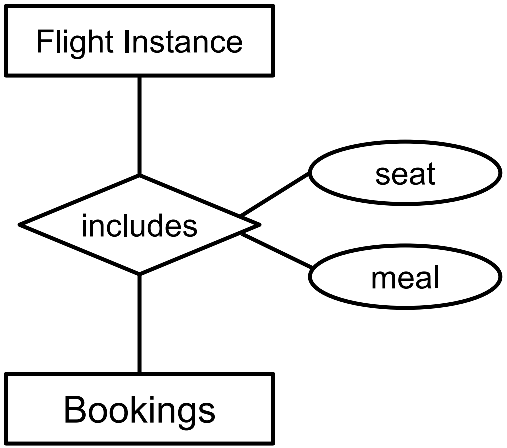
- Cardinalities of Relationship Sets
  - Describe how often an entity can participate in a relationship at most

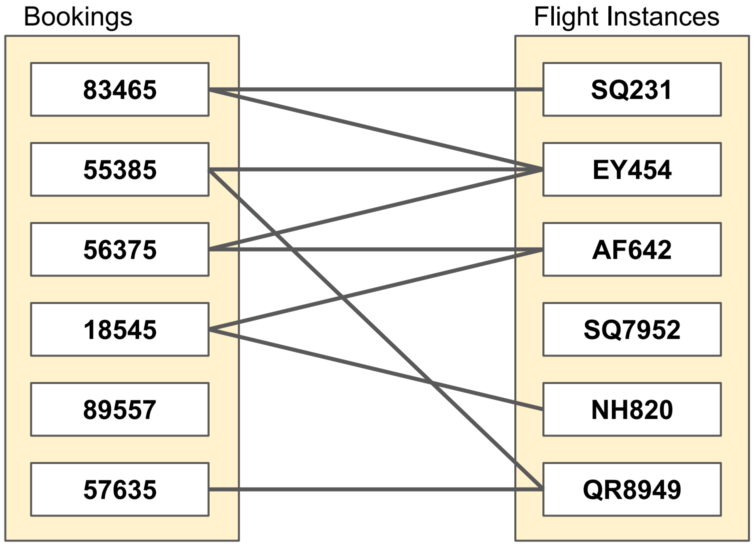
- 3 basic cardinality constraints
  - **Many-to-many** (e.g., a flight can be performed by different aircrafts; an aircraft can perform different flights)
  - **Many-to-one** (e.g., a user can make many bookings, but each booking is done by one user)
  - **One-to-one** (e.g., a user is associated with one set of credit card details, and vice versa)

- Cardinality constraints can be specified using annotations in ER diagram
  - Note: different ways to specify cardinality constraints available

Many-to-Many

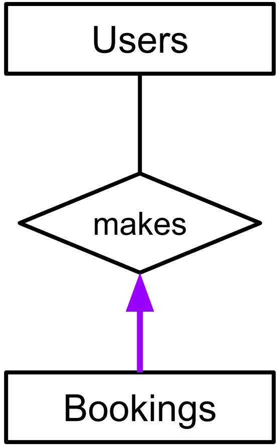
- Many-to-many relationship between bookings and flight instances
  - Each booking can include 0 or more flight instances
    (note that a booking with 0 flights might not meaningful; we will improve on that)
  - Each flight instance can be part of 0 or more bookings

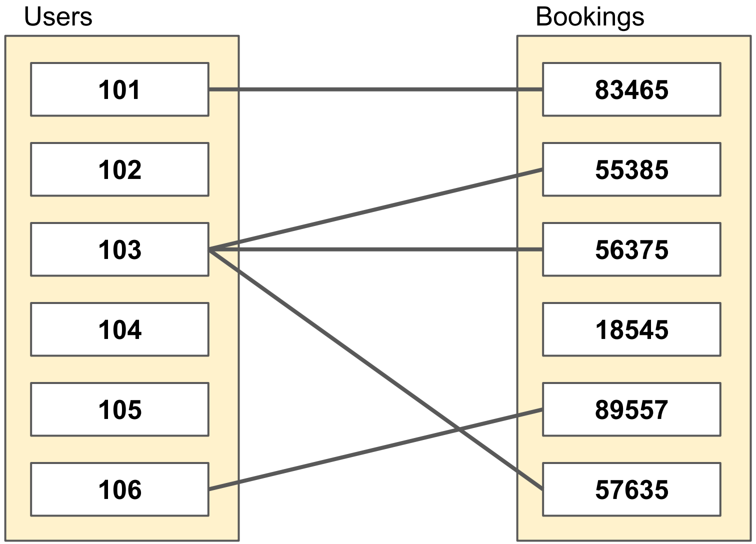




Many-to-One

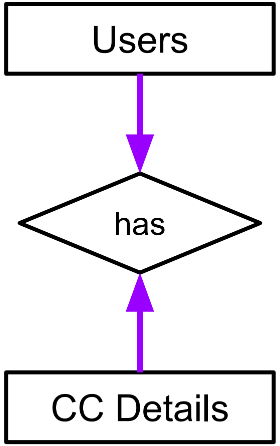
- Many-to-one relationship between users and bookings
  - Each user can make 0 or more bookings
  - Each booking is done by one 1 user at most
    (again, not perfect yet, and we will improve on that)

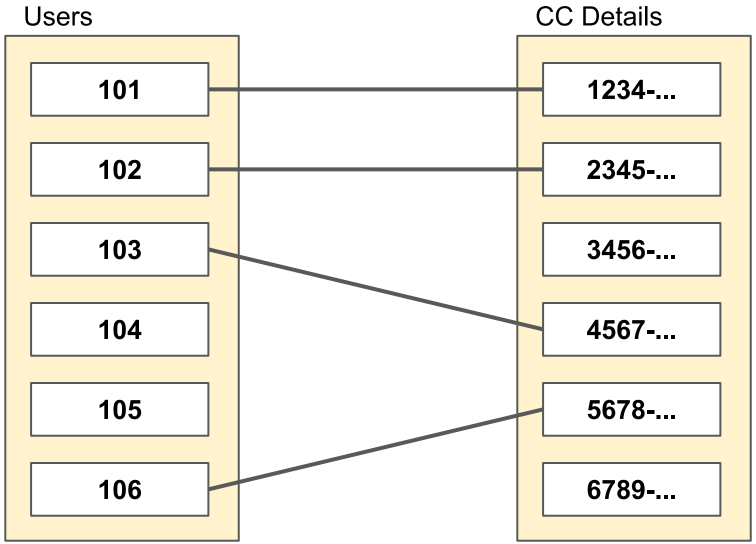




One-to-One

- One-to-one relationship between users and credit card details
  - Each user can provide only 1 set of credit card details at most
  - Each set of credit card details is associated with 1 user at most





Participation Constraints

- Limitation of (basic) cardinality constraints from previous examples
  - A booking can include 0 flights
  - A booking can be done by 0 users
  - A set of credit card details does not need to be associated with a user

an entity does not have to participate in a relation

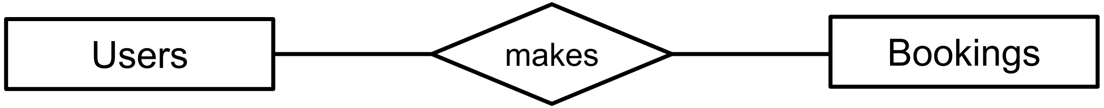
➜ Cardinality constraints (many-to-many, many-to-one, one-to-one) only specify some kind of "upper bound"

➜ **Participation constraints**

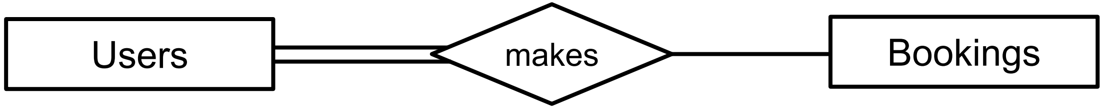
- Is the participation of an entity in a relationship mandatory?
- Allow to specify a trivial lower bound

Participation Constraints

- **Partial** participation constraint (default)
  - Participation of an entity in a relationship is not mandatory
  - Example: A user made 0 or more bookings

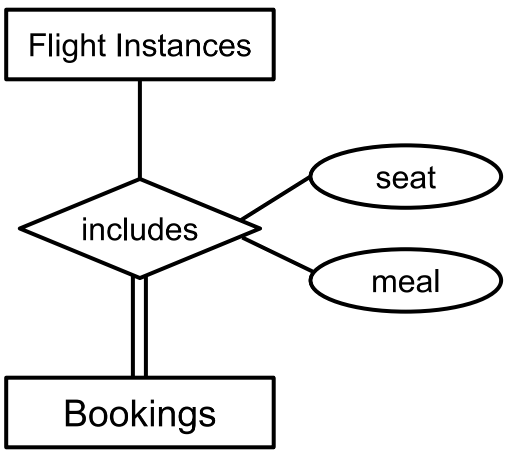


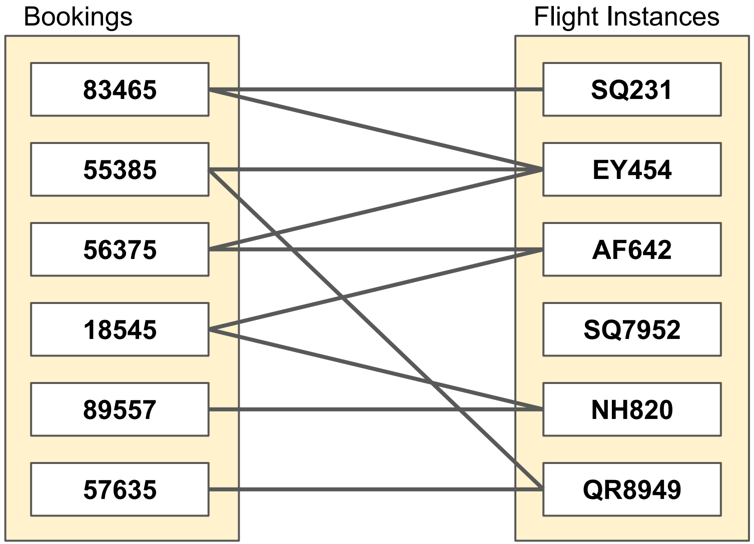
- Total participation constraint
  - Participation of an entity in a relationship is mandatory
  - Example: We only keep user that made at least one booking



Cardinality & Participation Constraints

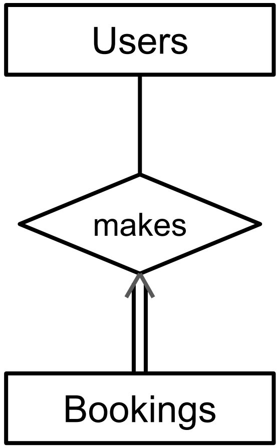
- Many-to-many relationship between bookings and flight instances
  - Each booking includes <u>1 or more</u> flight instances
  - Each flight instance can be part of 0 or more bookings

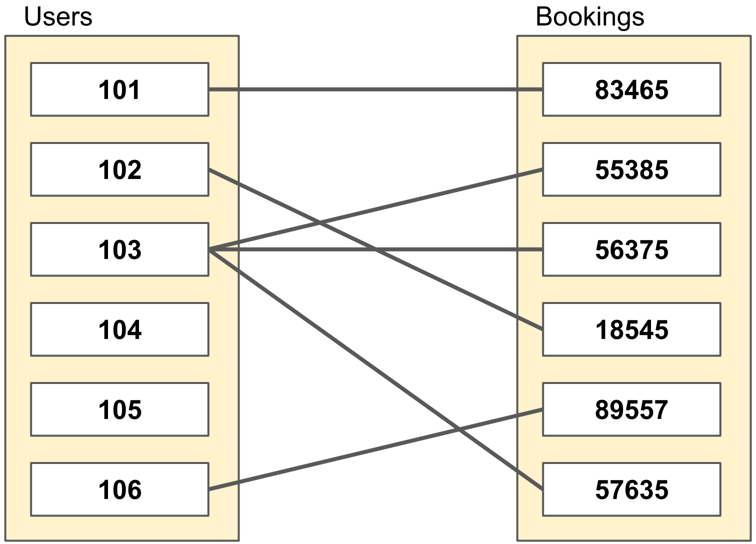




Cardinality & Participation Constraints

- Many-to-one relationship between users and bookings
  - Each user can make 0 or more bookings
  - Each booking is done by <u>exactly 1</u> user

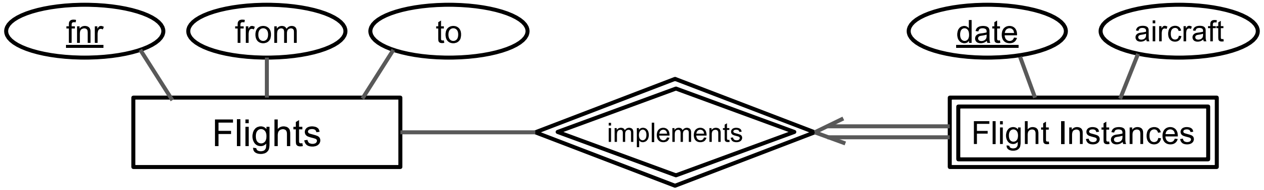




Dependency Constraints

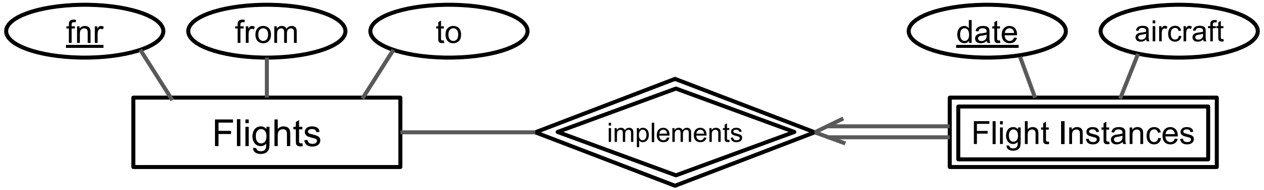
- **Weak entity sets**
  - Entity set that does not have its own key
  - A weak entity can only be uniquely identify by considering the primary key of the **owner entity**
  - A weak entity's existence depends on the existence of its owner entity
  - Weak entity set and identifying relation set are represented via double-lined rectangles / diamonds

- Example
  - A flight instance is the actual scheduled flight (with a unique flight number) on a given day

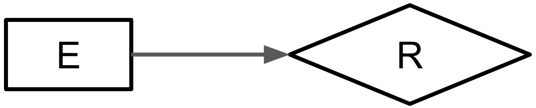


Dependency Constraints

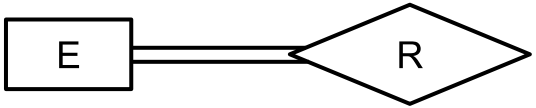
- Requirements
  - Many-to-one relationship (identifying relationship) from weak entity set to owner entity set
  - Weak entity set must have total participation in identifying relationship
- Partial key
  - Set of attributes of weak entity set that uniquely identifies a weak entity for a given owner entity
  - Example: Given a flight (e.g. SQ231), the date identifies the exact instance of that flight



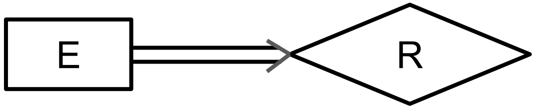
Summary of Participation Constraints



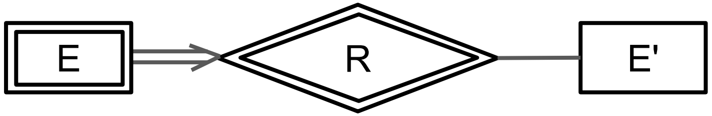
Each instance of E participates in at most one instance of R.



Each instance of E participates in at least one instance of R.



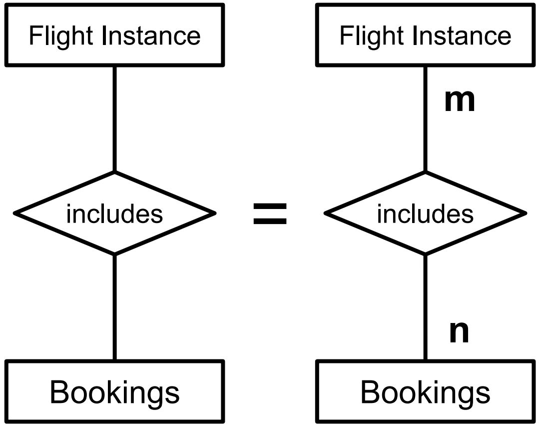
Each instance of E participates in exactly one instance of R.



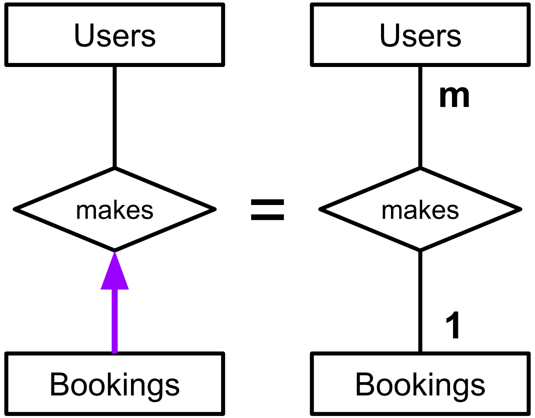
E is a weak entity set with identifying owner E' and identifying relationship set R.

Alternative Representations (Cardinality Constraints)

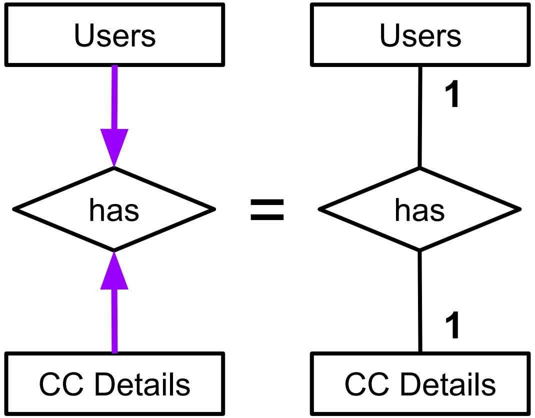
Many-to-Many



Many-to-One

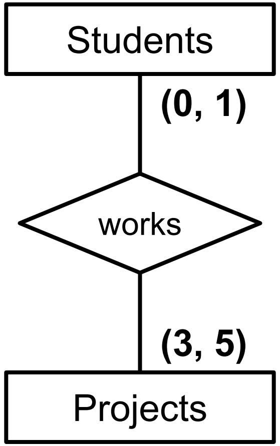


One-to-One



Alternative Representations (Cardinality Constraints)

- Min/Max notation
  - Specification of precise lower and upper bounds



A student works on exactly 1 project, or no project at all.

A project is assigned to teams comprising 3 to 5 students.

Quick Quiz: Why is this more precise notation in practice often not that useful?

Solution

- With what we've learned so far, we could not enforce these constraints
- At least not the (3, 5) constraint

Overview

- Entity Relationship Model
  - Overview + ER diagrams
  - Entity sets and attributes
  - Relationship sets
  - Cardinality & participation constraints

- **Relational Mapping**
  - From ER diagram to database tables

- Extended notations for ER diagrams
  - ISA hierarchies: generalization/specialization
  - Aggregation

Database Design Process — 6 Common Steps

Requirement Analysis

- Identification and collection of user needs
- e.g., data /application / performance requirements

Conceptual DB Design

- Capturing requirements using a conceptual model
- RDBMS: **Entity Relationship Model (ER Model)**

Logical DB Design

- Mapping conceptual model to logical schema of DBMS
- RDBMS: Entity Relationship Model → Relational Schema

Schema Refinement

- Checking schema / tables for redundancies and anomalies

Physical DB Design

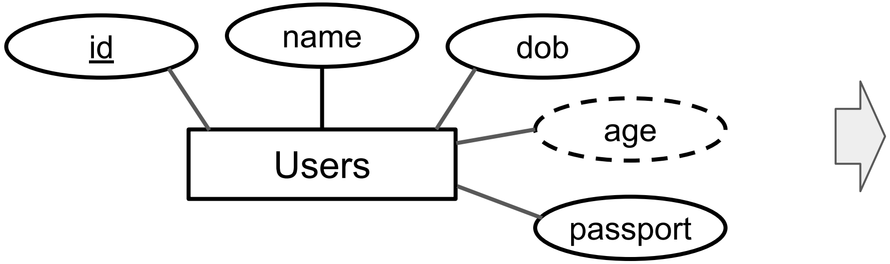
- Implementing database based on final data schema
- Consideration of performance requirements

Security Design

- Identification users and user groups and their permissions to access which parts of the data

Entity Sets

- Straightforward mapping from entity sets to tables (except composite & multivalued attributes)
  - Name of entity set ➜ name of table
  - Attributes of entity set ➜ attributes of table
  - Key attributes of entity set ➜ primary key of table

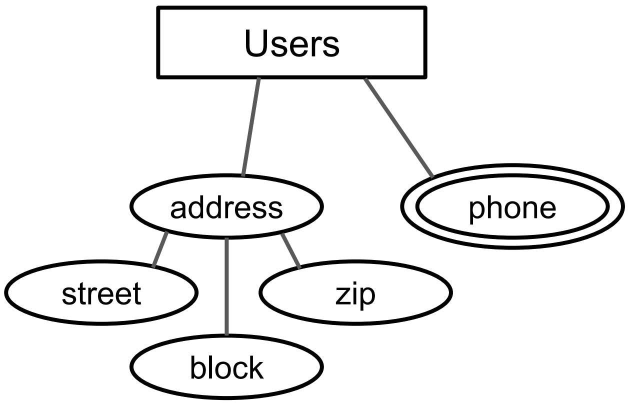



```
CREATE TABLE Users (
        id          INTEGER,
        name        VARCHAR(100),
        dob         DATE,
        age         INTEGER,
        passport    VARCHAR(20),
        PRIMARY KEY (id)
);
```

Note: The ER diagram does not specify UNIQUE or NOT NULL constraints that are potentially meaningful when creating a table.

Composite & Multivalued Attributes

- Problem: Tables can only hold atomic values
  (ignoring complex data types support by some DBMS)

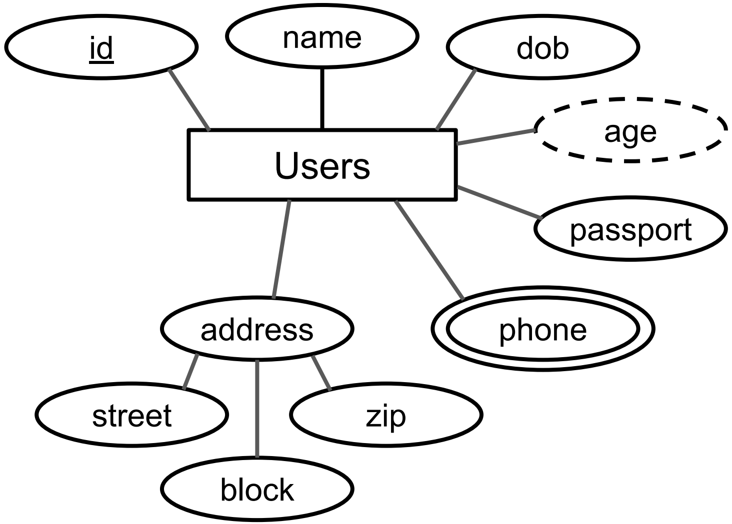


- 2 principle solutions + 1 alternative
  - Convert composite & multivalued attributes into a set of single-valued attributes
  - Create additional tables with a foreign key constraint referencing table of original entity set
    (typically only meaningful for multivalued attributes)
  - Convert composite & multivalued attributes to one single-valued attribute (if meaningful)

Note: One can design the ER diagram without composite and multivalued attributes using additional entity and relationship set which yield the same result as the proposed solutions.

Composite & Multivalued Attributes

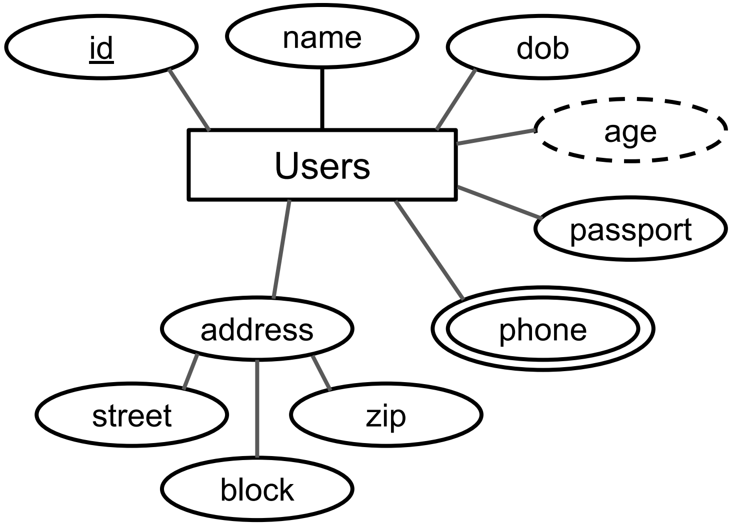
- Conversion to single-valued attributes
  - Requires an upper limit in case of multivalued attributes




```
CREATE TABLE Users (
        id          INTEGER,
        name        VARCHAR(100),
        dob         date,
        age         INTEGER,
        passport    VARCHAR(20),
        street      VARCHAR(50),
        block       VARCHAR(6),
        zip         INTEGER,
        phone1      INTEGER,
        phone2      INTEGER,
        phone3      INTEGER,
        PRIMARY KEY (id)
);
```

(Composite &) Multivalued Attributes

- Additional table with foreign key constraint

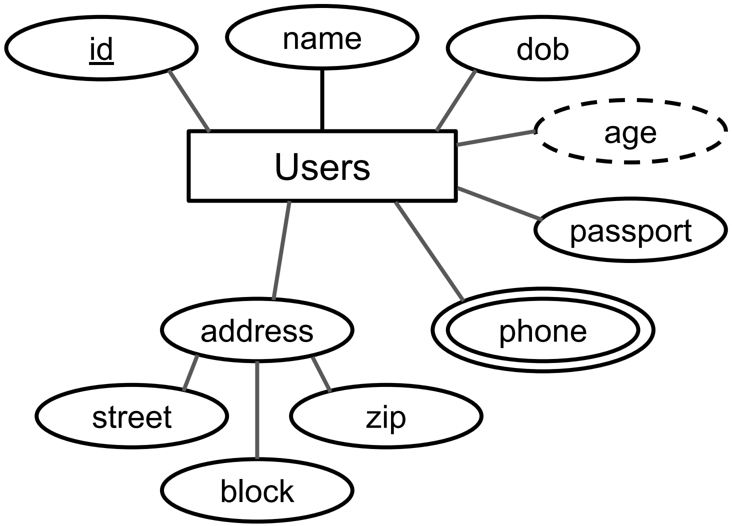



```
CREATE TABLE Users (
        id              INTEGER,
        name            VARCHAR(100),
        ...
        PRIMARY KEY (id)
);

CREATE TABLE PhoneNumbers (
        user_id         INTEGER,
        phone           INTEGER,
        ...
        FOREIGN KEY (user_id) REFERENCES Users (id)
);
```

Composite & Multivalued Attributes

Quick Quiz: What are the problems with this approach and when is it meaningful (enough)?

- Convert to single-valued attribute




```
CREATE TABLE Users (
        id              INTEGER,
        name            VARCHAR(100),
        dob             date,
        age             INTEGER,
        passport        VARCHAR(20),
        address         VARCHAR(200),
        phone           VARCHAR(200),
        PRIMARY KEY (id)
);
```

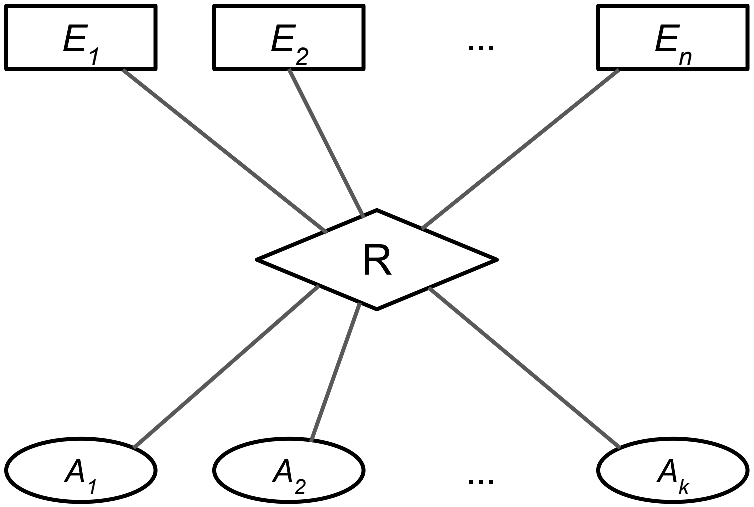
id	name	dob	age	passport	address	phone
101	Alice	15-02-2000	21	KEJR4A90	15 Computing Drive, Singapore 117418	65-1111-2222, 65-2222-3333, 65-3333-4444

Quick Quiz

Solution

- Modeling "address" and "phone" as a single-values string might be OK-ish if we never use these attributes to select rows
- If we only need to get the address or all phone numbers for a given user the this solution might be good enough
- However, queries using "address" or "phone" to filter rows will become unnecessarily complicated or even impossible
- A query such as "Return all users with addresses with the ZIP code 123456" are possible since SQL supports string pattern matching and even regular expression. The performance would degrade, though.
- More intricate queries might still be formulated but the complexity of the SQL query would quickly blow up

Relationship Sets



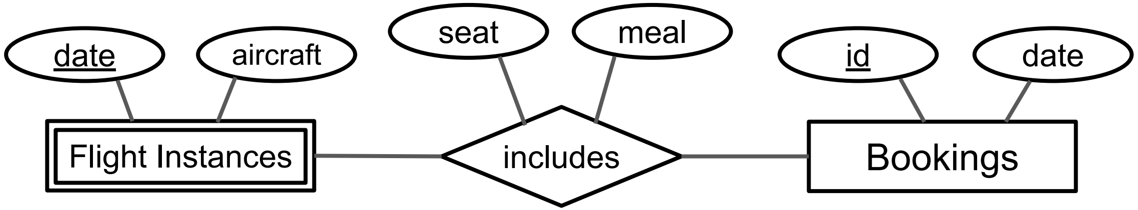
- General n-ary relationship set R
  - n participating entity sets $E_1, E_2, \ldots, E_n$
  - k relationship attributes $A_1, A_2, \ldots, A_k$
  - Let $Key(E_i)$ be the attributes of the selected key of entity set E_i

➜ **Attributes of relationship set *R***

- $Key(E_1), Key(E_2), \ldots, Key(E_n)$ — key attributes of all participating entity sets E_i
- $A_1, A_2, \ldots, A_k$ — all relationship attributes of R

Cardinality Constraints: Many-to-Many

Quick Quiz: Where does "flight_nr" come from?



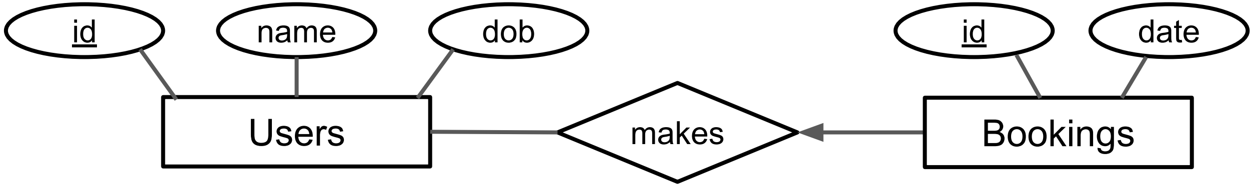

```
CREATE TABLE Includes (
        flight_nr       VARCHAR(10),
        flight_date     DATE,
        booking_id      INTEGER,
        seat            VARCHAR(10),
        meal            VARCHAR(50),
        PRIMARY KEY (flight_nr, flight_date, booking_id),
        FOREIGN KEY (flight_nr, date) REFERENCES FlightInstances (fnr, date),
        FOREIGN KEY (booking_id) REFERENCES Bookings (id),
);
```

Quick Quiz

Solution

- "Flight Instances" is weak entity set with "Flights" being the owner entity set
- Thus, "Flight Instances" is identified by the key of "Flights" (i.e., "fnr") and its own partial key "date"

Cardinality Constraints: Many-to-One

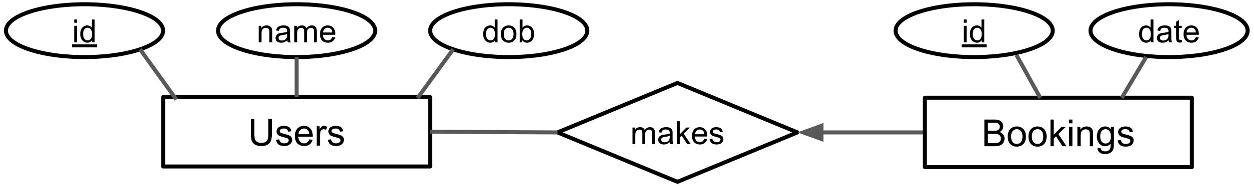


- **Approach 1**: Represent "makes" with a separate table
  - Similar to Many-to-Many but with different primary key!

```
CREATE TABLE Makes (
        user_id         INTEGER,
        booking_id      INTEGER,
        PRIMARY KEY (booking_id),
        FOREIGN KEY (user_id) REFERENCES Users (id),
        FOREIGN KEY (booking_id) REFERENCES Bookings (id)
);
```

Cardinality Constraints: Many-to-One

Quick Quiz: Which is generally the preferred approach?



- **Approach 2**: Combine "makes" and "Bookings" into one table
  - Possible because given a booking, we can uniquely identify the user who made it

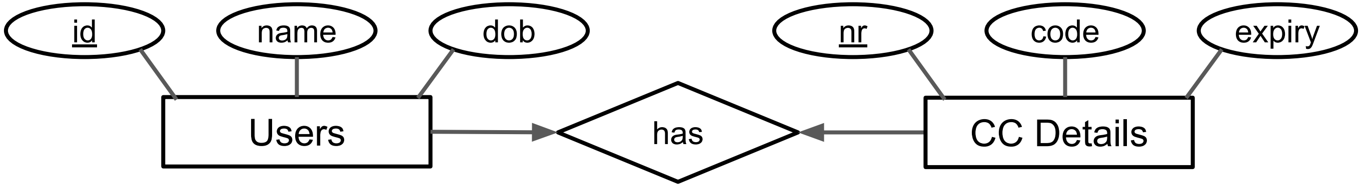
```
CREATE TABLE Bookings (
        id              INTEGER,
        date            DATE,
        user_id         INTEGER,
        PRIMARY KEY (id),
        FOREIGN KEY (user_id) REFERENCES Users (id)
);
```

Quick Quiz

Solution

- Approach 2 is generally the preferred approach as it leads to a smaller number of table
- Less tables also means that queries might need less join operations
  (which are typically the more expensive operations).

Cardinality Constraints: One-to-One

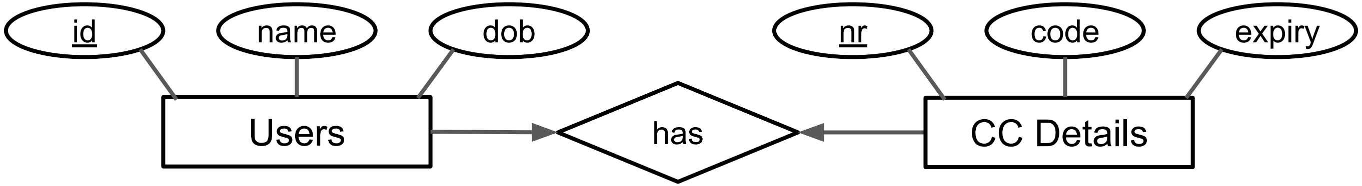


- **Approach 1**: Represent "has" with a separate table
    - Similar to Many-to-One but primary key can be chosen

```
CREATE TABLE Has (
        user_id         INTEGER,
        cc_nr           CHAR(16) UNIQUE,
        PRIMARY KEY (user_id),
        FOREIGN KEY (user_id) REFERENCES Users (id),
        FOREIGN KEY (cc_nr) REFERENCES CCDetails (id)
);
```

or vice versa!

Cardinality Constraints: One-to-One



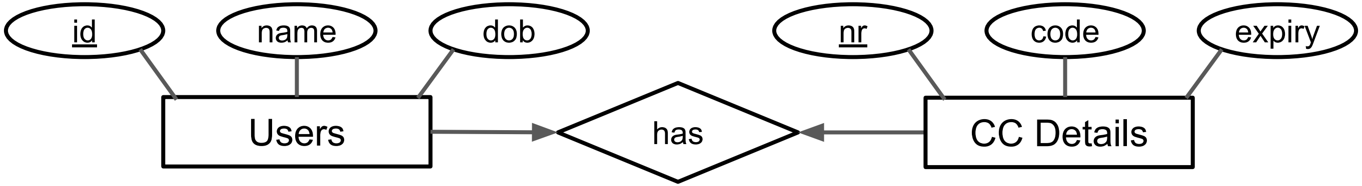
- **Approach 2**: Combine "has" and "Users" or "has" and "CC Details"

```
CREATE TABLE Users (
    id          INTEGER,
    name        VARCHAR(100),
    dob         DATE,
    cc_nr       CHAR(16),
    PRIMARY KEY (id),
    FOREIGN KEY (cc_nr) REFERENCES CCDetails (nr)
);
```

OR

```
CREATE TABLE CCdetails (
    nr          CHAR(16),
    code        CHAR(3),
    expiry      DATE,
    user_id     INTEGER,
    PRIMARY KEY (nr),
    FOREIGN KEY (user_id) REFERENCES Users (id)
);
```

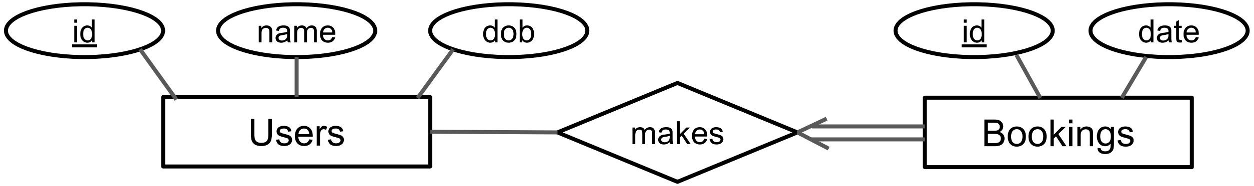
Cardinality Constraints: One-to-One



- **Approach 3**: Combine "has", "Users", and "CC Details"

```
CREATE TABLE Users (
        id              INTEGER,
        name            VARCHAR(100),
        dob             DATE,
        cc_nr           CHAR(16) UNIQUE,
        cc_code         CHAR(3),
        cc_expiry       DATE,
        PRIMARY KEY (id)
);
```

Cardinality & Participation Constraints

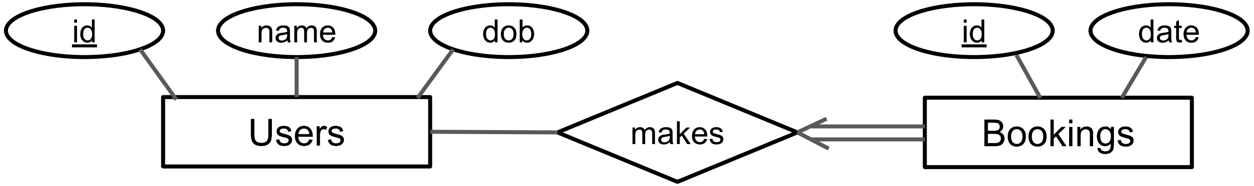


- **Approach 1**: Represent "makes" with a separate table

```
CREATE TABLE Makes (
        user_id         INTEGER NOT NULL,
        booking_id      INTEGER,
        PRIMARY KEY (booking_id),
        FOREIGN KEY (user_id) REFERENCES Users (id),
        FOREIGN KEY (booking_id) REFERENCES Bookings (id)
);
```

- Schema does not enforce total participation of "Bookings" w.r.t. "Makes"
- e.g.: "Makes" can be empty while both "Users" and "Bookings" are non-empty

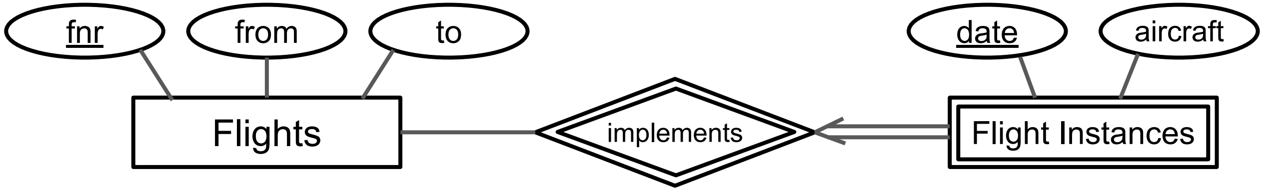
Cardinality & Participation Constraints



- **Approach 2**: Combine "makes" and "Bookings" into one table
  - Enforces total participation via NOT NULL constraint

```
CREATE TABLE Bookings (
        id              INTEGER,
        date            DATE,
        user_id         INTEGER NOT NULL,
        PRIMARY KEY (id),
        FOREIGN KEY (user_id) REFERENCES Users (id)
);
```

Weak Entity Sets




```
CREATE TABLE Flights (
        fnr             VARCHAR(10),
        from            VARCHAR(10),
        to              VARCHAR(10),
        PRIMARY KEY (fnr)
);
```

```
CREATE TABLE FlightInstances (
        fnr             VARCHAR(10),
        date            DATE,
        aircraft        VARCHAR(10),
        PRIMARY KEY (fnr, date),
        FOREIGN KEY (fnr) REFERENCES Flights (fnr)
                ON DELETE CASCADE
);
```

ER Design & Relational Mapping — Basic Guidelines

- Guidelines for ER design
  - An ER diagram should capture as many of the constraints as possible
  - An ER diagram must not impose any constraints that are not required

- Guidelines for relational mapping
  (i.e., from ER diagram to relational database schema)
  - The relational schema should enforce as many if the constraints as possible using column and/or table constraints
  - The relational schema should not impose and constraints that are not required

Overview

- Entity Relationship Model
  - Overview + ER diagrams
  - Entity sets and attributes
  - Relationship sets
  - Cardinality & participation constraints

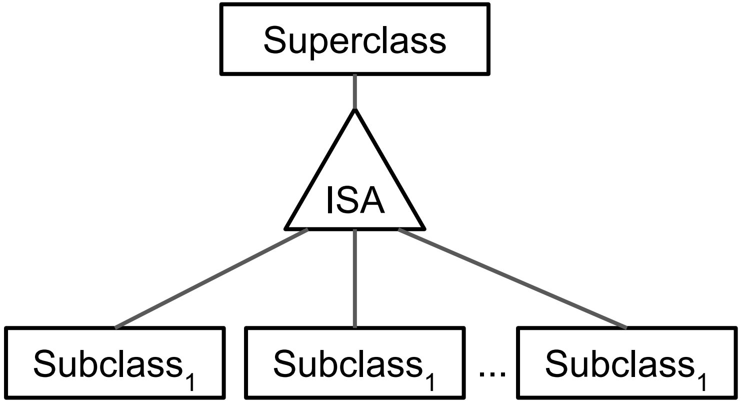
- Relational Mapping
  - From ER diagram to database tables

- **Extended notations for ER diagrams**
  - ISA hierarchies: generalization/specialization
  - Aggregation

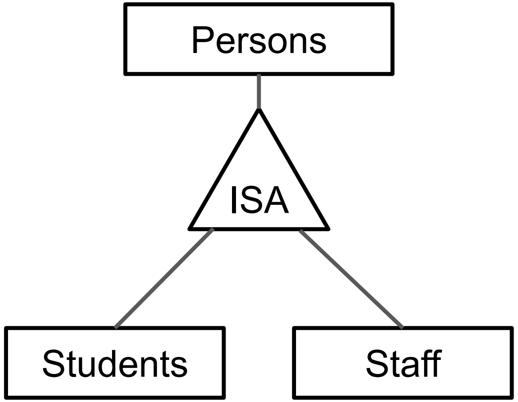
Extended Concepts — ISA Hierarchies

- ISA hierarchies
  - Special type of relationship: "is a"
  - Used to model generalization/specialization of entity sets

General representation

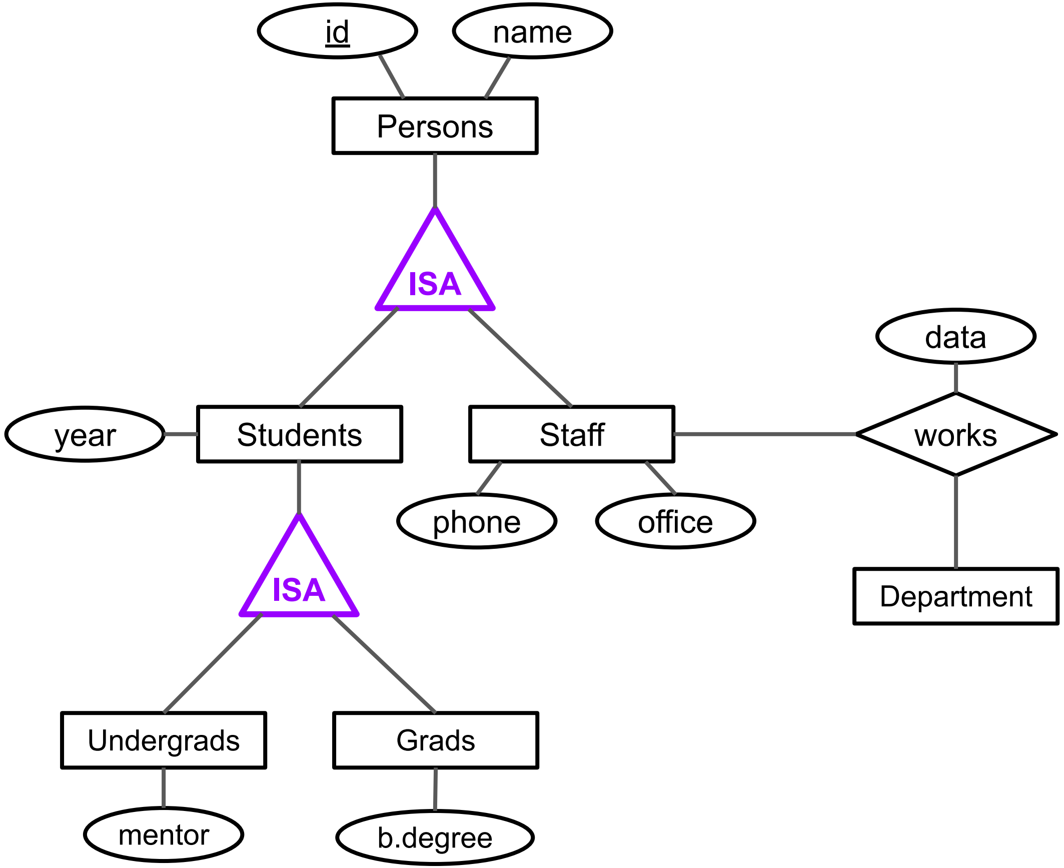


Example



ISA Hierarchies

- Interpretation
  - Every entity in a subclass is an entity in its superclass
  - Each subclass has specific attributes and/or relationships



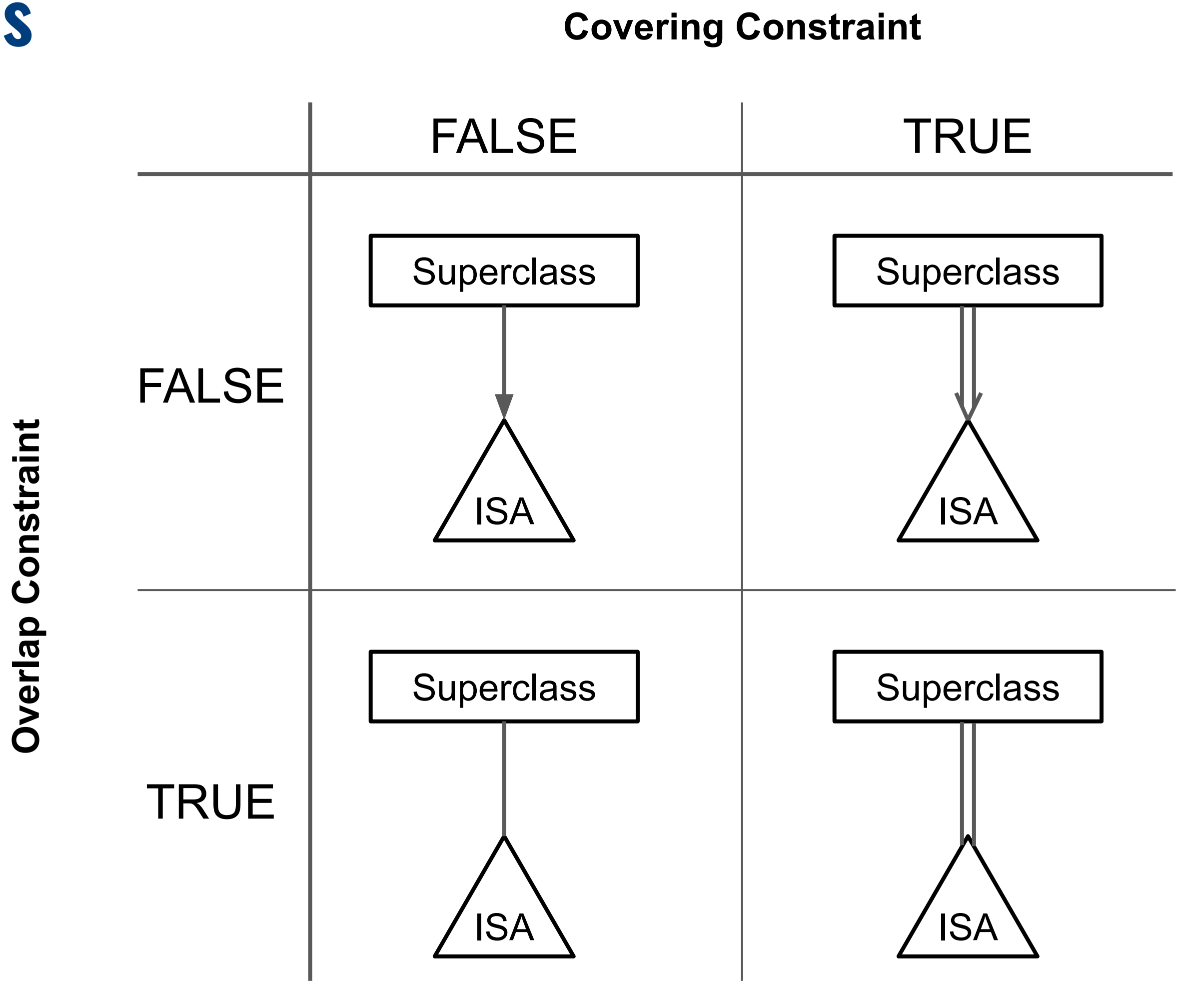
ISA Hierarchies — Constraints

- **Overlap constraint:** Can a superclass entity belong to multiple subclasses?
  - TRUE ➜ a superclass entity can belong to multiple subclasses
    (e.g., a person can be both student and staff)
  - FALSE ➜ otherwise
    (e.g., a student is either a graduate or undergraduate)

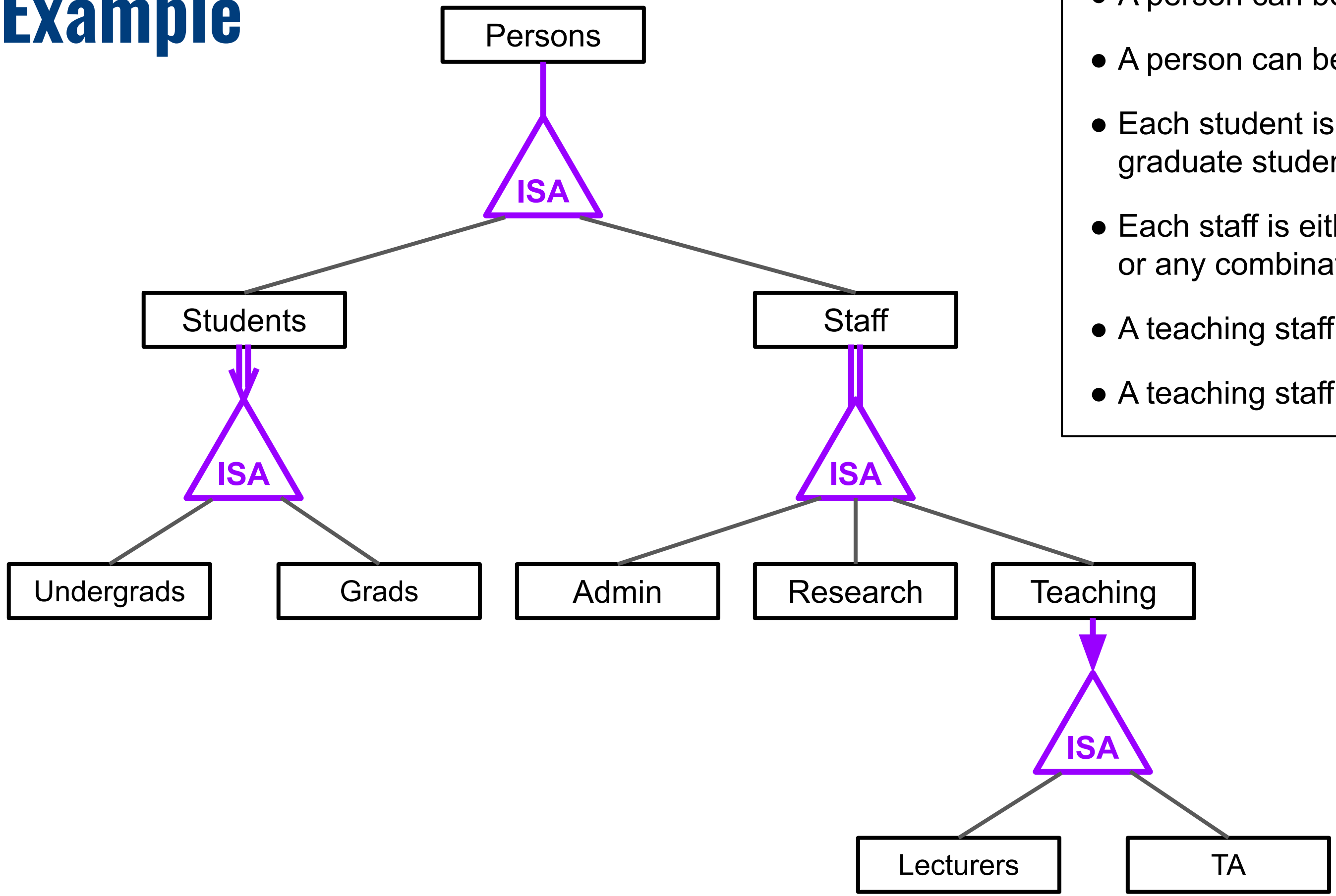
- **Covering constraint:** Does a superclass entity have to belong to a subclass?
  - TRUE ➜ every superclass entity has to belong to a subclass
    (e.g., there is no student that is neither a graduate or undergraduate)
  - FALSE ➜ otherwise
    (e.g., not every person is a student or staff)

ISA Hierarchies

Notation in ER Diagram



Example

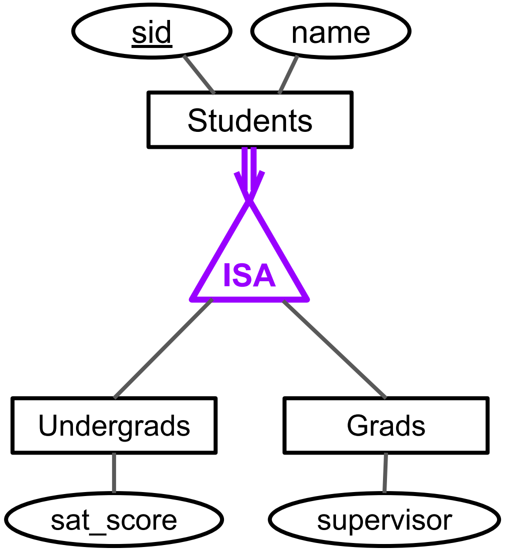


- A person can be both a student and staff
- A person can be neither a student or staff
- Each student is either a undergraduate or a graduate student (but definitely one of both)
- Each staff is either *admin*, *research*, *teaching* or any combination of these three roles
- A teaching staff cannot be a lecturer and TA
- A teaching staff can be neither lecturer or TA

ISA Hierarchies: Relational Mapping

- Basic approach: One relation per subclass and superclass

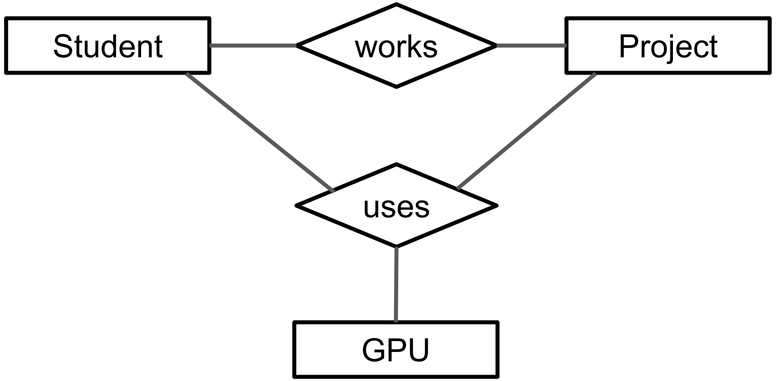
```
CREATE TABLE Students (
        sid             CHAR(20) PRIMARY KEY,
        name            VARCHAR(50)
);

CREATE TABLE Undergrads (
        sid             CHAR(20) PRIMARY KEY,
        sat_score       NUMERIC,
        FOREIGN KEY (sid) REFERENCES Students (sid) ON DELETE CASCADE
);

CREATE TABLE Grads (
        sid             CHAR(20) PRIMARY KEY,
        supervisor      CHAR(8),
        FOREIGN KEY (sid) REFERENCES Students (sid) ON DELETE CASCADE,
        FOREIGN KEY (supervisor) REFERENCES Staff (id) ON DELETE SET NULL
);
```



Extended Concepts — Aggregation

- Concepts of ER diagrams so far
  - Only relationships between entity sets
  - No relationships between entity sets and relationship sets

- Motivational example



Limitations:

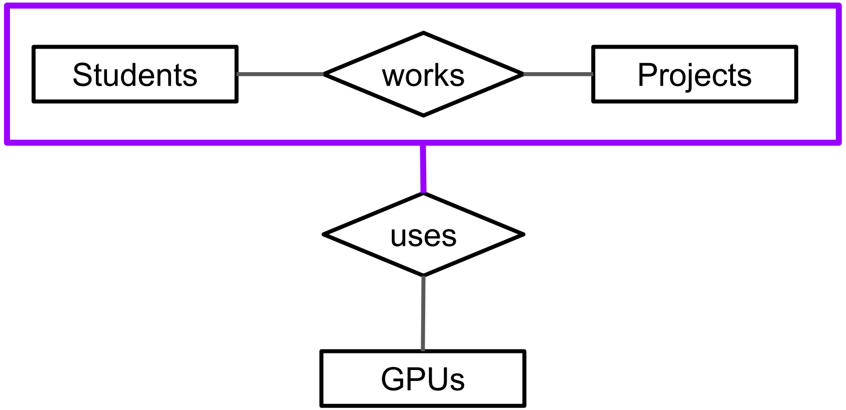
- Relationship between "works" and "uses" not explicitly captured
- "works" and "uses" are kind of redundant relationships

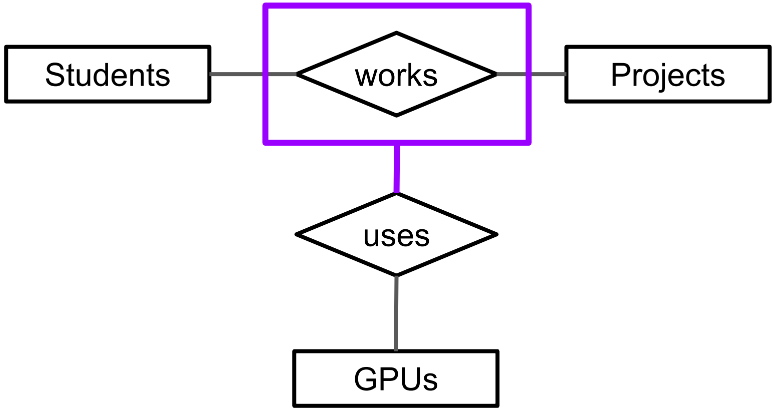
➜ **Aggregation**

Extended Concepts — Aggregation

- Aggregation — basic idea
  - Abstraction that treats relationships as higher-level entities
  - Example: treat Student-works-Project as an entity set

- Notation in ER diagram (2 equivalent alternatives)

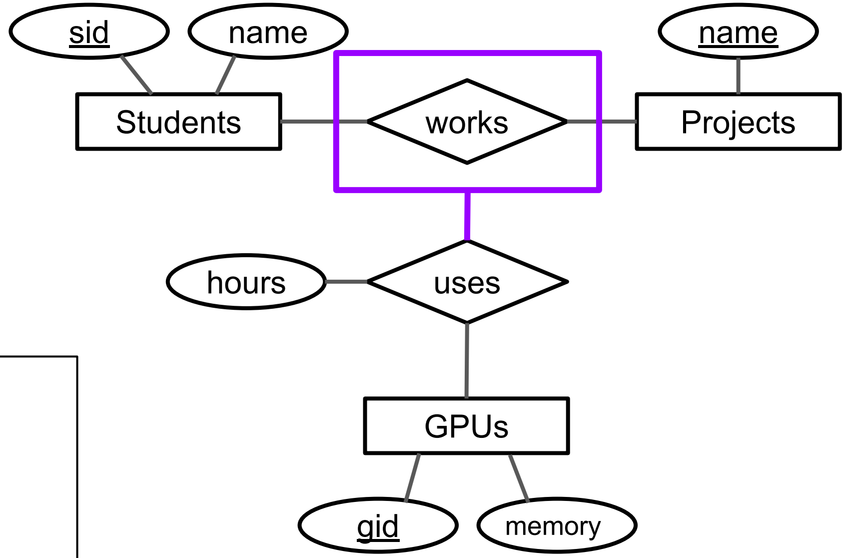




Aggregation — Relational Mapping

Schema definition of "uses"

- Primary key of aggregation relationship ➜ (sid, pname)
- Primary key of associated entity set "GPUs" ➜ gid
- Descriptive attributes of "uses" ➜ hours




```
CREATE TABLE Uses (
        gid             INTEGER,
        sid             CHAR(20),
        pname           VARCHAR(50),
        hours           NUMERIC,
        PRIMARY KEY (gid, sid, pname),
        FOREIGN KEY (gid) REFERENCES GPUs (gid),
        FOREIGN KEY (sid, pname) REFERENCES works (sid, pname)
);
```

Summary

- Entity-Relationship (ER) model
  - Basic concepts: entity sets, relationship sets, attributes
  - Cardinality constraints and participation constraints
  - Extended concepts: ISA hierarchies, aggregation

Visualized using **ER diagrams**

- Relational Mapping
  - Mapping ER diagram to database schema
  - Not all constraints of ER diagram may be captured

- Outlook for next lecture
  - SQL for querying a database (recommendation: study RA)