Mapping ER to SQL (And Constraints)

R&G Chapter 3, 5

(slides adapted from content by J.Gehrke, J.Shanmugasundaram, and/or C.Koch)

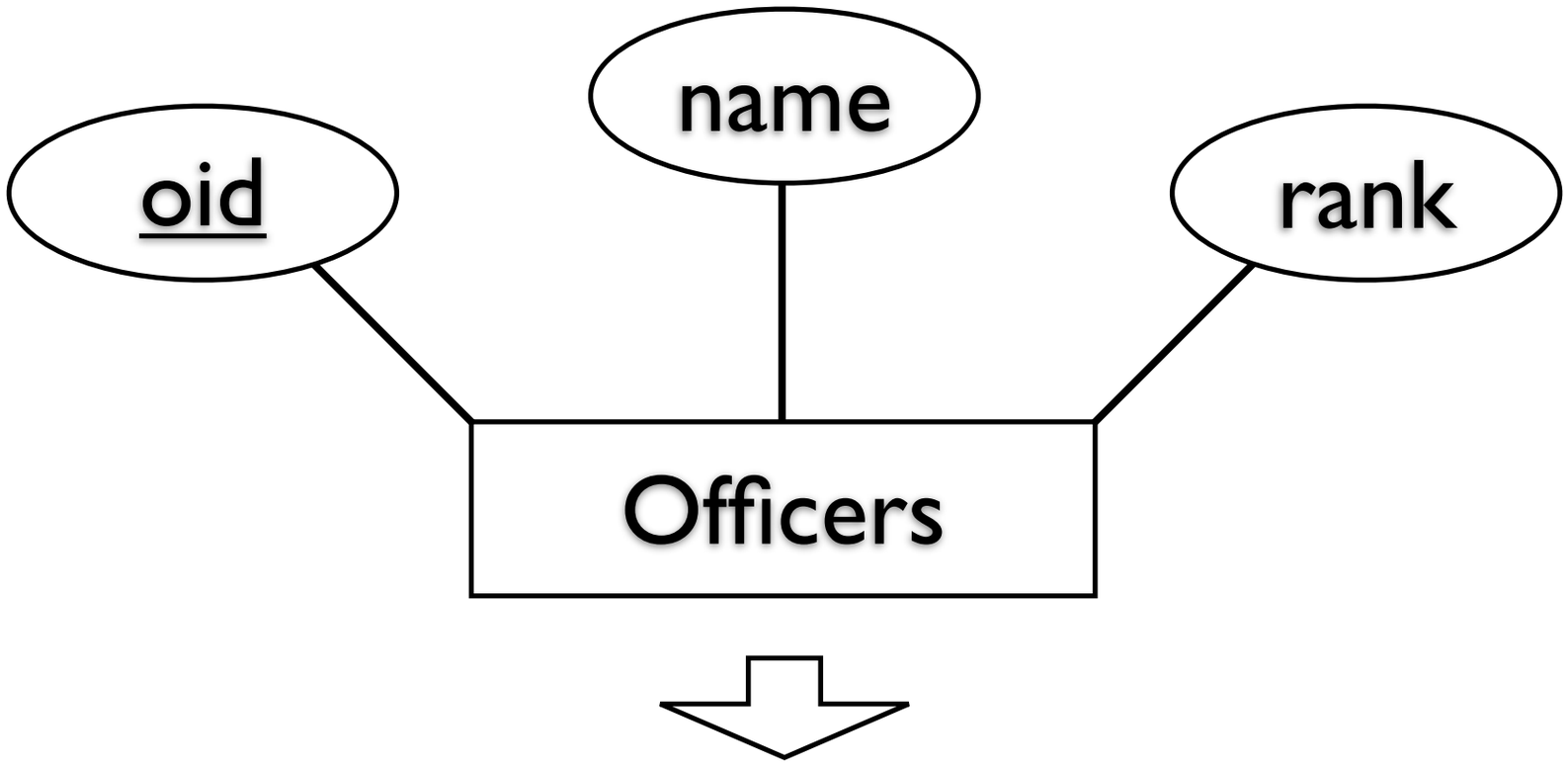
Recap: Data Modeling

- The ER Model is a popular way to design schemas (and maps nicely to SQL)
  - **Basic Constructs**: Entities, Relationships, and Sets of both. (Sets equivalent to SQL Relations)
  - **Additional Constructs**: Weak Entities, ISA hierarchies, Aggregation
- There is no one 'right' model for a given scenario.
- Understanding how to design a schema is important.

Recap: Keys/Constraints

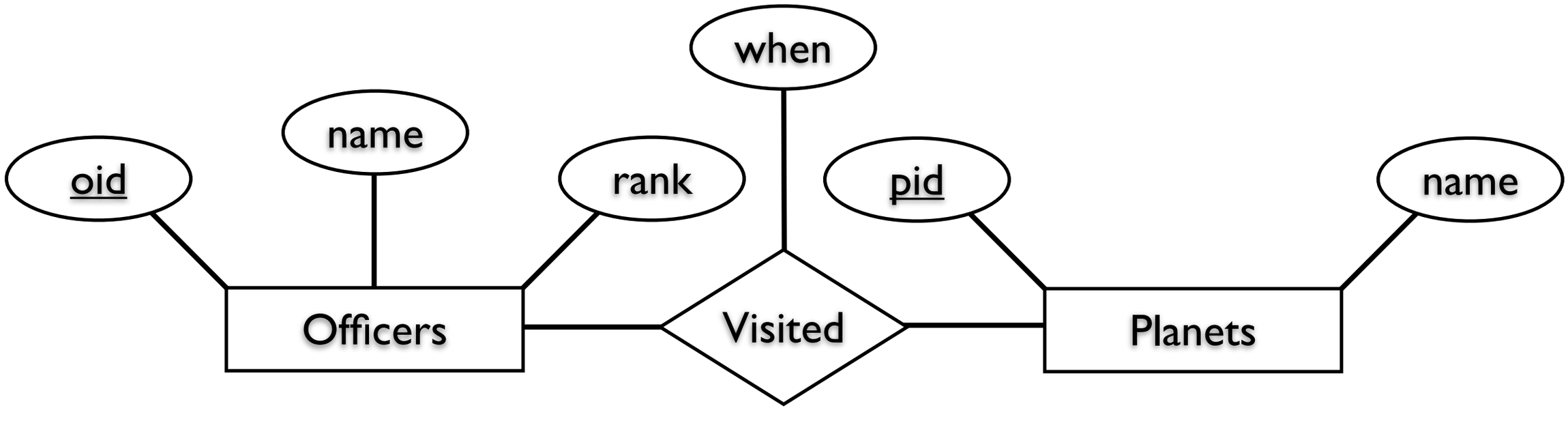
- All entities have a key attribute.
- All relationships join pairs of entities.
  - Participation constraints require an entity to have at least one relationship.
    - One-To-Many/Many-To-One/One-To-One relationships restrict entity participation.
- Weak entities are defined by their relationships.

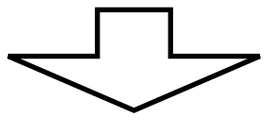
Basics




```
CREATE TABLE Officers(
  oid INTEGER,
  name CHAR(50),
  rank REAL
);
```

Basics





```
CREATE TABLE Visited(
   oid INTEGER,
   pid INTEGER,
   when DATE
);
```

Integrity Constraints

- "Correctness" Properties on Relations
  - … enforced by the DBMS.
- Typically simple uniqueness/existence properties, paralleled by ER Constraints
  - … we'll discuss more complex properties when we discuss Triggers later in the term.
- Database optimizers benefit from constraints.

Integrity Constraints

- Domain Constraints
  - Limitations on valid values of a field.
- Key Constraints
  - A field(s) that must be unique for each row.
- Foreign Key Constraints
  - A field referencing a key of another relation.
  - Can also encode participation/1-many/many-1/1-1.
- Table Constraints
  - More general constraints based on queries.

Domain Constraints

- Stronger restrictions on the contents of a field than provided by the field's type
  - e.g., 0 < Rank ≤ 5
- Mostly present to prevent data-entry errors.

Postgres:

```
CREATE DOMAIN Rank AS REAL
   CHECK (0 < VALUE AND VALUE <= 5)
```

Oracle:

```
CREATE TABLE Officers (
   …
   Rank REAL,
   CHECK (0 < Rank AND Rank <= 5) );
```


Thursday, February 14, 13

There's no standardized mechanism for defining domain restrictions
Postgres defines domain restrictions through the type system. You define a new type (analogous to typedef in C, or subclassing in Java), which can have restrictions on legitimate values.
Oracle defines domain restrictions through its (far more general, as we'll soon see) table constraint mechanism.

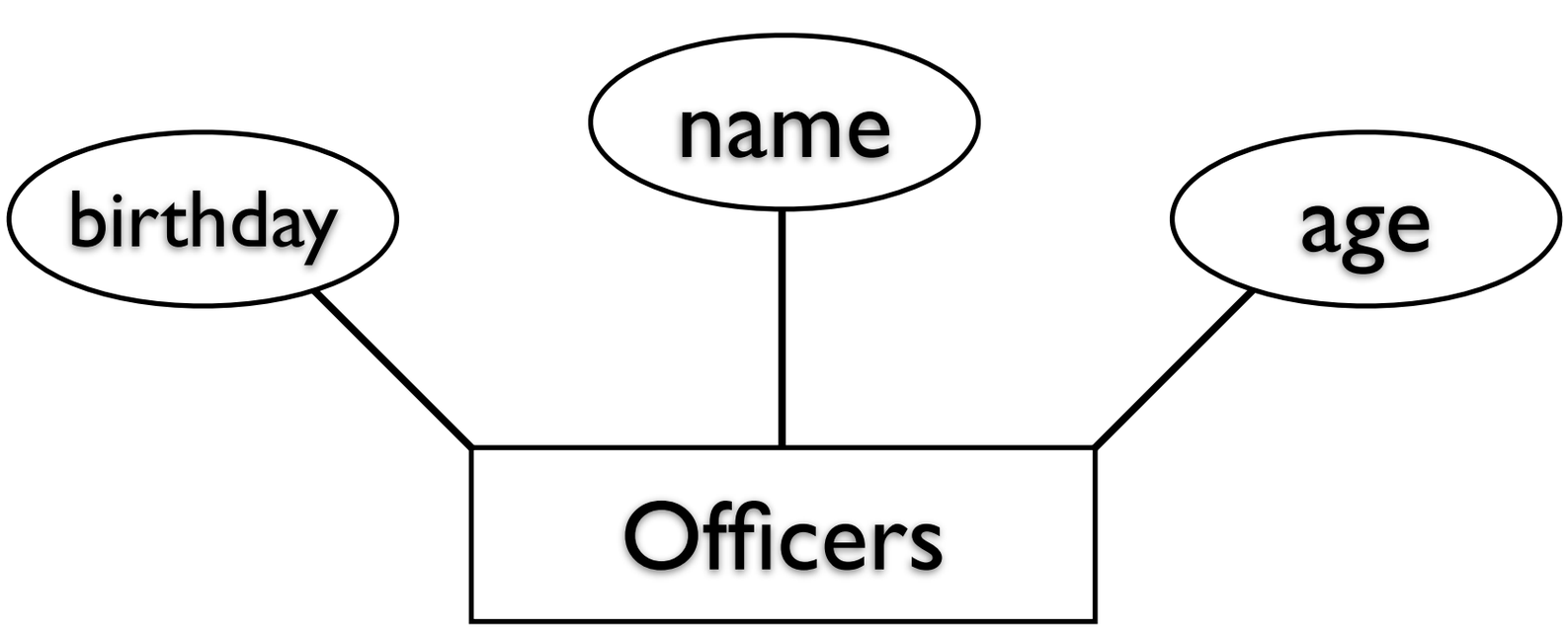
Domain Constraints

- Special domain constraint: NOT NULL
  - Field not allowed to contain NULL values.

```
CREATE TABLE Officer(
  oid INTEGER NOT NULL,
  name CHAR(50),
  birthday DATE
);
```

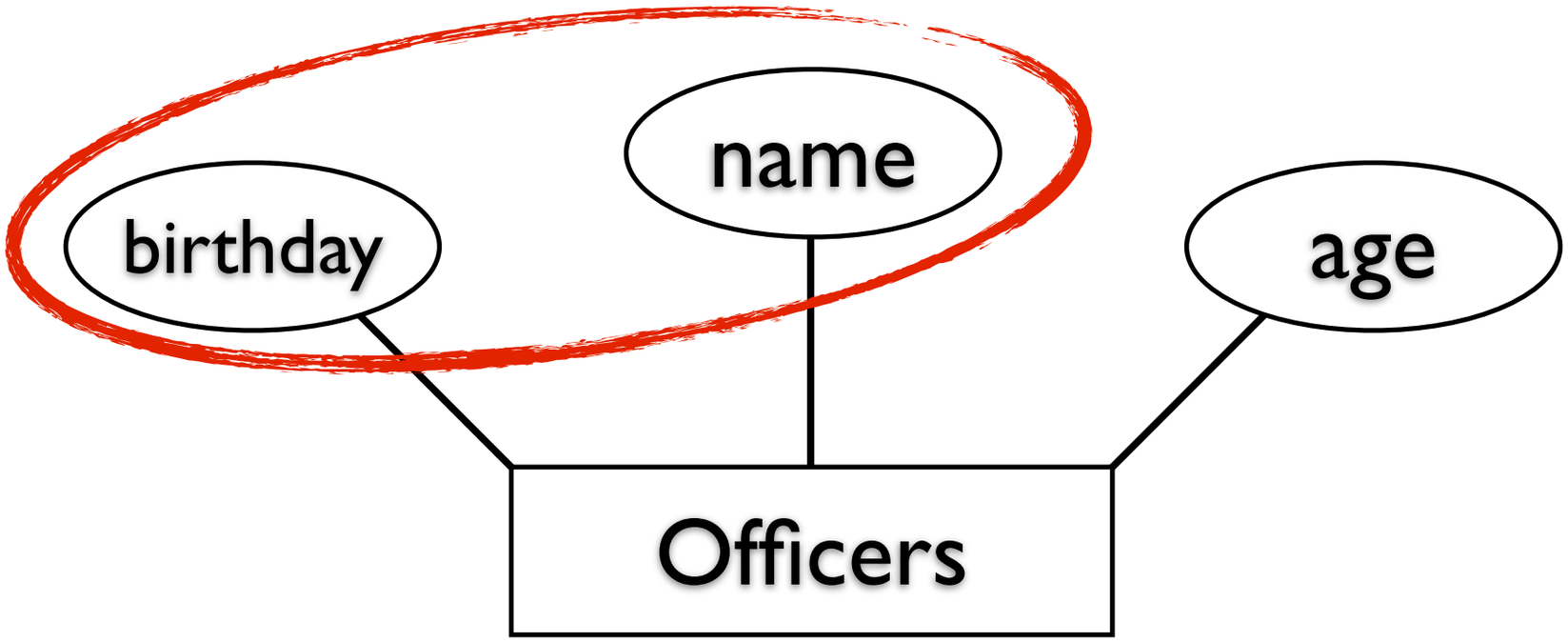
Key Constraints

- A set of fields that uniquely identifies a tuple in a relation.
- There can be multiple keys for a relation.



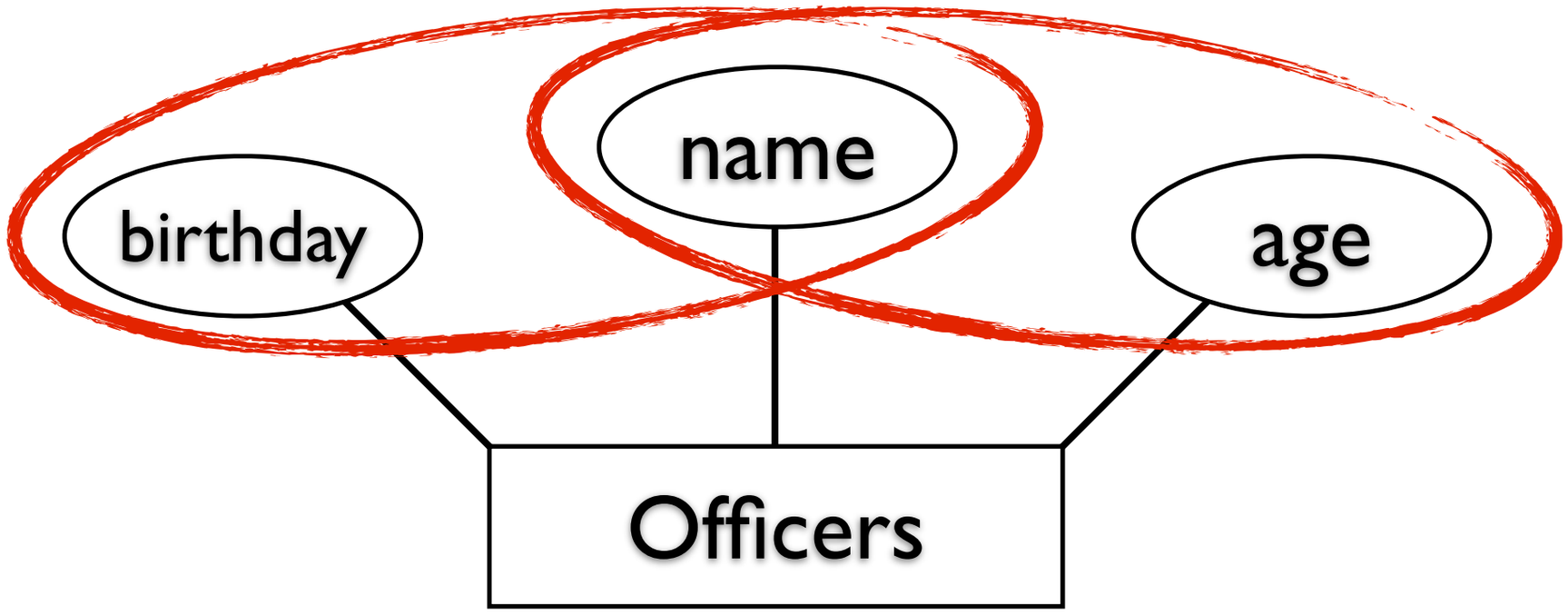
Key Constraints

- A set of fields that uniquely identifies a tuple in a relation.
- There can be multiple keys for a relation.



Key Constraints

- A set of fields that uniquely identifies a tuple in a relation.
- There can be multiple keys for a relation.

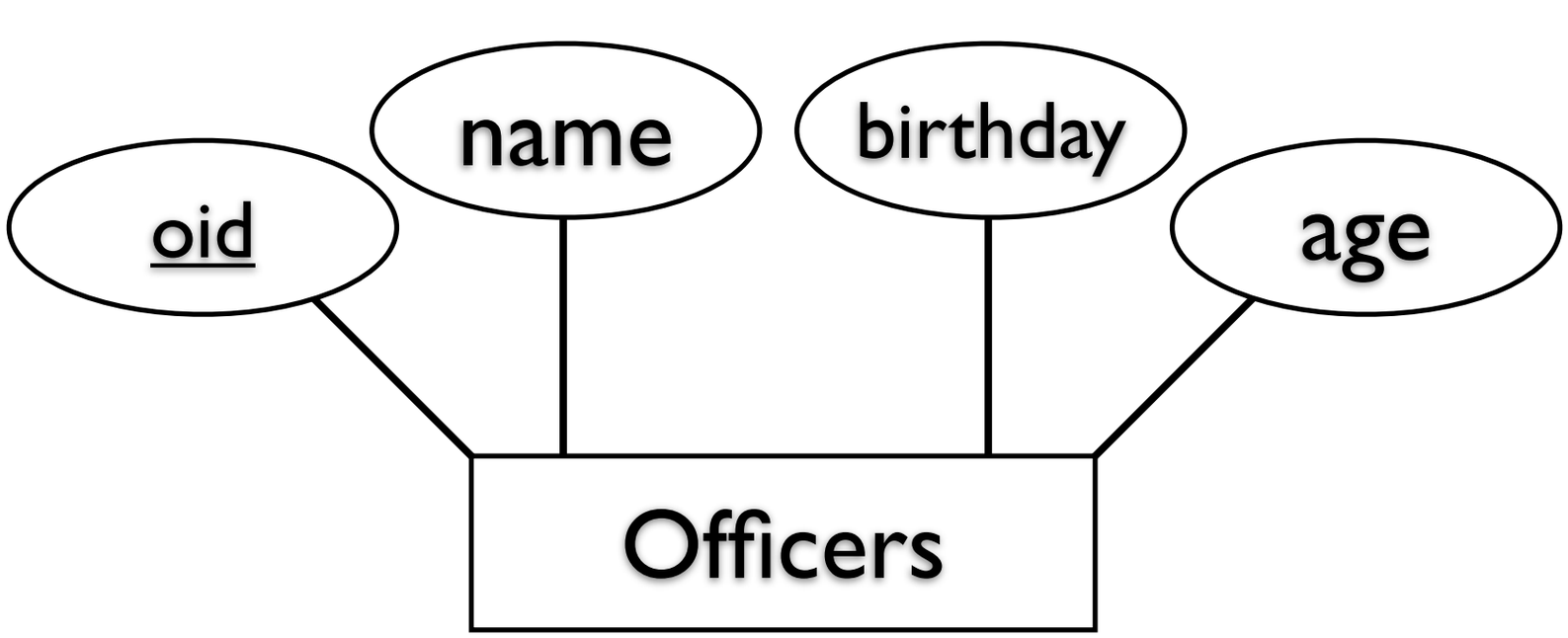


Key Constraints

- A *key* satisfies the following two properties:
  - No two distinct tuples have identical values in all the fields of a key.
    - Two officers can have the same name, or the same birthday/age, but not both name and birthday/age.
  - No subset of the fields of a key has the above property.
    - Name+Age+Birthday is not a key (it is a *superkey*)
    - Name+Age is a key, and Name+Birthday is a key.

Defining Key Constraints

```
CREATE TABLE Officer(
  oid INTEGER, name CHAR(50),
  birthday DATE, age REAL,
  UNIQUE (name, age),
  CONSTRAINT OfficerDay UNIQUE (name, birthday),
  PRIMARY KEY (oid)
);
```



Defining Key Constraints

```
CREATE TABLE Officer(
  oid INTEGER, name CHAR(50),
  birthday DATE, age REAL,
  UNIQUE (name, age),
  CONSTRAINT OfficerDay UNIQUE (name, birthday),
  PRIMARY KEY (oid)
);
```

UNIQUE identifies a key constraint

Defining Key Constraints

```
CREATE TABLE Officer(
   oid INTEGER, name CHAR(50),
   birthday DATE, age REAL,
   UNIQUE (name, age),
   CONSTRAINT OfficerDay UNIQUE (name, birthday),
   PRIMARY KEY (oid)
);
```

UNIQUE identifies a key constraint

Defining Key Constraints

```
CREATE TABLE Officer(
  oid INTEGER, name CHAR(50),
  birthday DATE, age REAL,
  UNIQUE (name, age),
  CONSTRAINT OfficerDay UNIQUE (name, birthday),
  PRIMARY KEY (oid)
);
```

UNIQUE identifies a key constraint

PRIMARY KEY identifies a key constraint that will commonly be used to refer to tuples in this relation.

Primary keys are typically used to make physical layout decisions (e.g., what key should we index on, or how should we sort the data), while UNIQUE is primarily just used to ensure correctness on the data. Primary keys are also not allowed to be NULL.

Defining Key Constraints

```
CREATE TABLE Officer(
   oid INTEGER, name CHAR(50),
   birthday DATE, age REAL,
   UNIQUE (name, age),
   CONSTRAINT OfficerDay UNIQUE (name, birthday),
   PRIMARY KEY (oid)
);
```

UNIQUE identifies a key constraint

PRIMARY KEY identifies a key constraint that will commonly be used to refer to tuples in this relation.

Thursday, February 14, 13

Primary keys are typically used to make physical layout decisions (e.g., what key should we index on, or how should we sort the data), while UNIQUE is primarily just used to ensure correctness on the data. Primary keys are also not allowed to be NULL.

Defining Key Constraints

```
CREATE TABLE Officer(
  oid INTEGER, name CHAR(50),
  birthday DATE, age REAL,
  UNIQUE (name, age),
  CONSTRAINT OfficerDay UNIQUE (name, birthday),
  PRIMARY KEY (oid)
);
```

UNIQUE identifies a key constraint

PRIMARY KEY identifies a key constraint that will commonly be used to refer to tuples in this relation.

CONSTRAINT (optionally) assigns a name to any constraint.

If the constraint is violated, DBMS error messages typically reference the constraint name, making errors easier to track down. The constraint name can also be used when modifying the relation later on to remove/modify the constraint.

Defining Key Constraints

```
CREATE TABLE Officer(
   oid INTEGER, name CHAR(50),
   birthday DATE, age REAL,
   UNIQUE (name, age),
   CONSTRAINT OfficerDay UNIQUE (name, birthday),
   PRIMARY KEY (oid)
);
```

UNIQUE identifies a key constraint

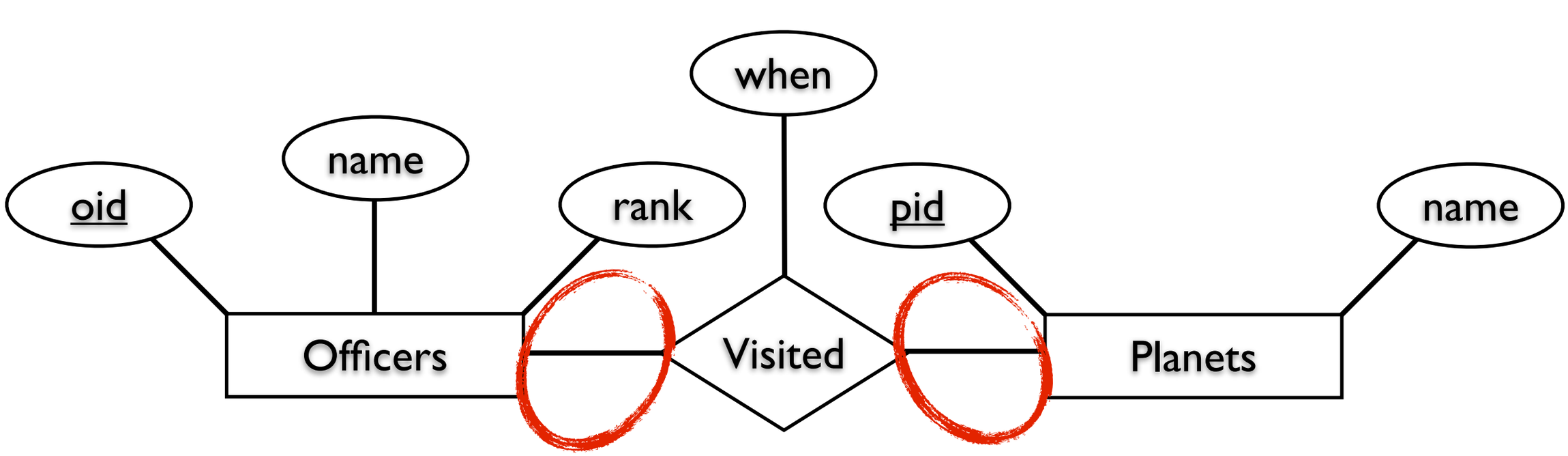
PRIMARY KEY identifies a key constraint that will commonly be used to refer to tuples in this relation.

CONSTRAINT (optionally) assigns a name to any constraint.

If the constraint is violated, DBMS error messages typically reference the constraint name, making errors easier to track down. The constraint name can also be used when modifying the relation later on to remove/modify the constraint.

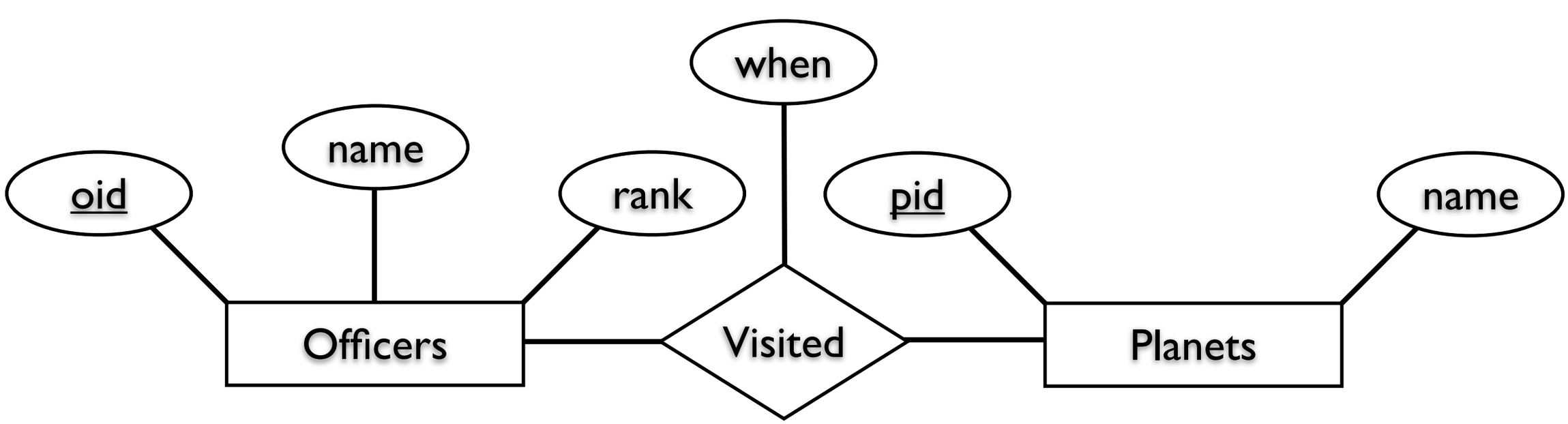
Foreign Key Constraints

- Used when a tuple in one relation needs to refer to a tuple in a different relation.
- The referenced tuple must exist.



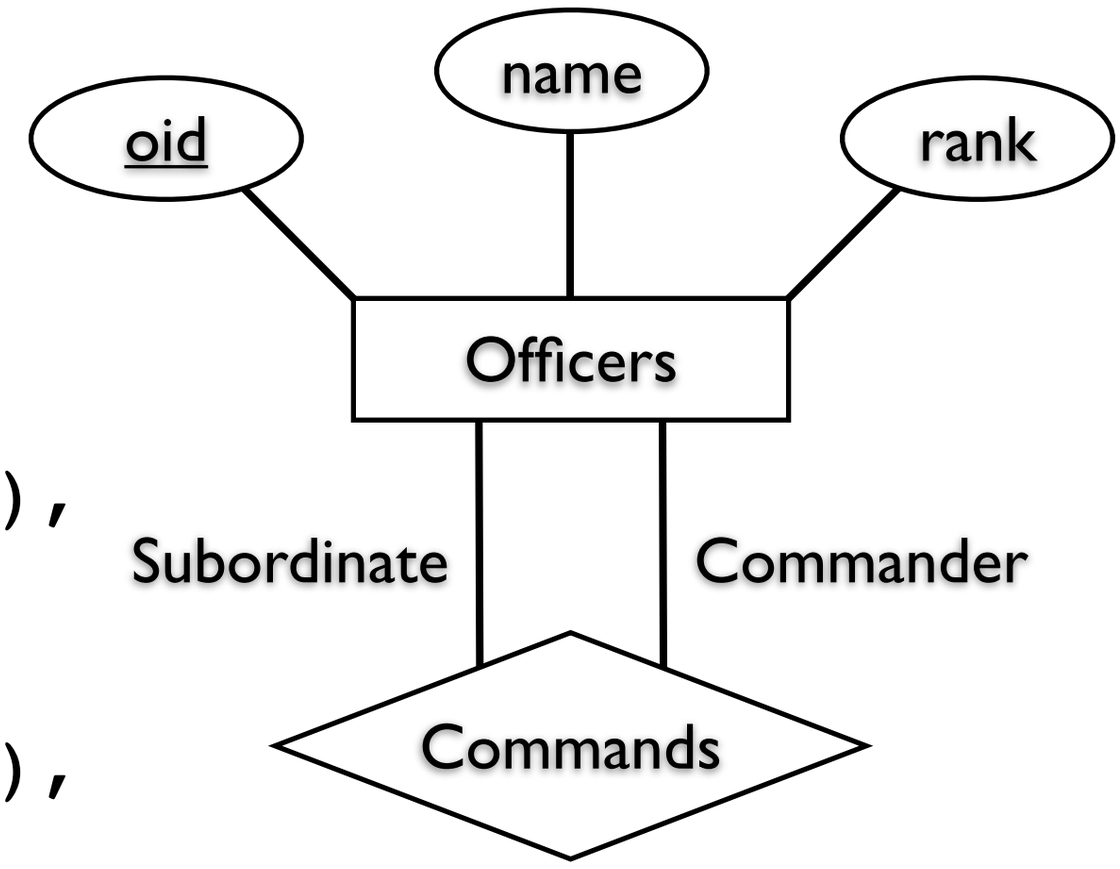
Foreign Key Constraints

```
CREATE TABLE Visited(
   oid INTEGER, pid INTEGER, when DATE,
   PRIMARY KEY (oid, pid),
   FOREIGN KEY (oid) REFERENCES Officers,
   FOREIGN KEY (pid) REFERENCES Planets
);
```



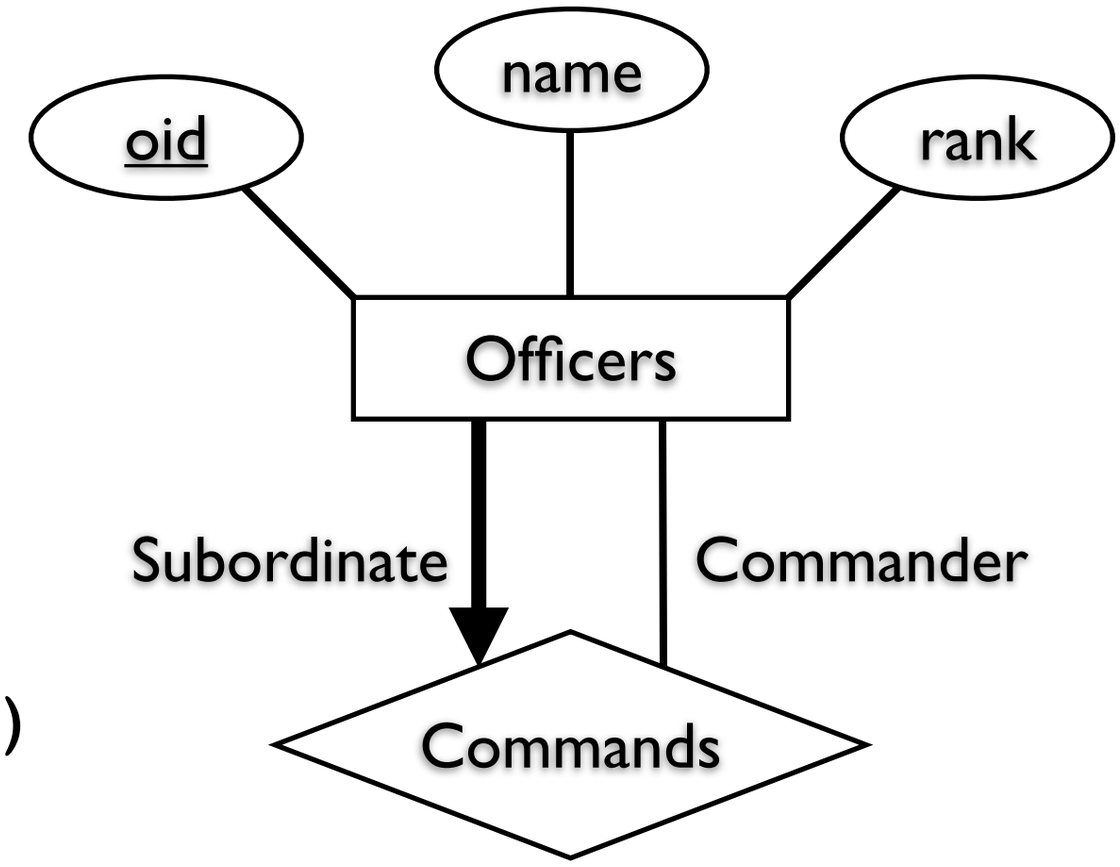
Foreign Key Constraints

```
CREATE TABLE Commands (
  Subordinate INTEGER,
  Commander INTEGER,

  PRIMARY KEY
    (Subordinate, Commander),

  FOREIGN KEY (Subordinate)
    REFERENCES Officers(oid),
  FOREIGN KEY (Commander)
    REFERENCES Officers(oid)
);
```



Foreign Key Constraints

```
CREATE TABLE Officers (
   …
   Commander INTEGER,
   …
   FOREIGN KEY (Commander)
     REFERENCES Officers(oid)
);
```



What about the Fleet Admiral (no commander)?
How do we insert the first tuple into Officers?

Thursday, February 14, 13

Foreign keys are allowed to be NULL values (which refer to no tuples)

Enforcing Constraints

- Basic Enforcement
  - Reject Inserts/Deletions/Updates that introduce constraint violations.
- Insertions: Domain, Key, FK Constraints
- Updates: Domain, Key, FK Constraints
- Deletions: Only FK Constraints

Referential Integrity Enforcement

- Foreign Key Constraints are complex
  - DBMSes will attempt to rectify violations rather than reject the violating update.
- How should we react to an inserted tuple that references a nonexistent foreign key?
- How should we react to a referenced tuple being deleted?
- How should we react to a referenced tuple being updated?

Referential Integrity Enforcement

How should we react to an inserted tuple that references a nonexistent foreign tuple?

Referential Integrity Enforcement

How should we react to an inserted tuple that references a nonexistent foreign tuple?

REJECT

Referential Integrity Enforcement

How should we react to a referenced tuple being deleted? (Delete Planet)

Note that primary key fields are not allowed to be NULL, so option 3 can often not be used.

Referential Integrity Enforcement

How should we react to a referenced tuple being deleted? (Delete Planet)

1. Delete all referencing tuples (Visited)

Note that primary key fields are not allowed to be NULL, so option 3 can often not be used.

Referential Integrity Enforcement

How should we react to a referenced tuple being deleted? (Delete Planet)

1. Delete all referencing tuples (Visited)
2. Disallow the deletion until there are no referencing tuples

Note that primary key fields are not allowed to be NULL, so option 3 can often not be used.

Referential Integrity Enforcement

How should we react to a referenced tuple being deleted? (Delete Planet)

1. Delete all referencing tuples (Visited)
2. Disallow the deletion until there are no referencing tuples
3. Replace the referencing foreign key by some default value (or NULL).

Note that primary key fields are not allowed to be NULL, so option 3 can often not be used.

Referential Integrity Enforcement

How should we react to a referenced tuple being updated? (Planet.pid changes)

Referential Integrity Enforcement

How should we react to a referenced tuple being updated? (Planet.pid changes)

1. Update all referencing tuples (change Visited.pid)

Referential Integrity Enforcement

How should we react to a referenced tuple being updated? (Planet.pid changes)

1. Update all referencing tuples (change Visited.pid)
2. Disallow the update until there are no referencing tuples

Referential Integrity Enforcement

How should we react to a referenced tuple being updated? (Planet.pid changes)

1. Update all referencing tuples (change Visited.pid)
2. Disallow the update until there are no referencing tuples
3. Replace the referencing foreign key by some default value (or NULL).

Referential Integrity Enforcement

```
CREATE TABLE Visited(
  oid INTEGER, pid INTEGER, when DATE,
  PRIMARY KEY (oid, pid),
  …
  FOREIGN KEY (pid) REFERENCES Planets
    ON DELETE CASCADE
    ON UPDATE NO ACTION
);
```

`CASCADE`	Delete or Update Reference
`NO ACTION`	Reject Deletion or Update
`SET DEFAULT v` `SET NULL`	Replace Reference with *v* or *NULL*

SET DEFAULT is almost always a bad idea.

Constraint Validation

- A *Transaction* is a batch of DBMS Operations
- `SET CONSTRAINT [name] IMMEDIATE;`
  - Perform constraint checking immediately after an insert/update/delete.
- `SET CONSTRAINT [name] DEFERRED;`
  - Perform constraint checking at the end of a transaction (commit time).

Thursday, February 14, 13

Deferred constraint checking provides two benefits

- For large transactions (e.g., bulk loading), it can be more efficient to validate constraints in one huge pass than to attempt re-validation on every operation

- Transactions are allowed to temporarily violate the constraint (e.g., by replacing one student record with another), as long as the constraint is once again satisfied by the end of the transaction.

Table Constraints

```
CREATE TABLE Officer(
  oid INTEGER,
  name CHAR(50),
  ship CHAR(5)
  PRIMARY KEY (oid)
  FOREIGN KEY (ship) REFERENCES Ships(sid)
  CHECK ( 'Enterprise' <> (SELECT Name
                           FROM Ship S
                           WHERE S.sid = Officer.ship))
);
```

CHECK clause can contain any conditional expression
If the conditional evaluates to false, the command is rejected

Some DBMSes place restrictions on what kind of conditions can appear in the check clause for efficiency's sake. Consult your local DBMS documentation when in doubt.

Multi-Table Constraints

Keep the number of Planets and Space Stations Over 100

```
CREATE TABLE SpaceStations (
  …
  CHECK ( 100 > (SELECT COUNT(*) FROM Planets)
                +(SELECT COUNT(*) FROM SpaceStations))
);
```


Problem 1: Design Flaw! The constraint is over both Planets and SpaceStations, but it's associated with Planets.

Problem 2: Constraints are only checked when the table is non-empty. If we have no space stations, we could lose the rest of our planets and still satisfy the constraint!

Multi-Table Constraints

Keep the number of Planets and Space Stations Over 100

```
CREATE ASSERTION SaveTheFederation
  CHECK ( 100 > (SELECT COUNT(*) FROM Planets)
                +(SELECT COUNT(*) FROM SpaceStations))
```

ASSERTION defines a CHECK that is not associated with any specific table.

Problem 1: Design Flaw! The constraint is over both Planets and SpaceStations, but it's associated with Planets.

Problem 2: Constraints are only checked when the table is non-empty. If we have no space stations, we could lose the rest of our planets and still satisfy the constraint!