The Relational Model
ER to Relational / SQL

Announcements

- Waiting list: all done (sorry)
- Project 1: Due Thursday morning
  - Late submissions: Under Eugene Wu's door (421 Mudd in the DSI space)

Review

Relation: Set of tuples with typed values (table)

Schema: Names and types for values in relation

Database: Set of relations

SQL: Structured Query Language

Integrity constraint: Restrictions on valid data

Candidate key: Minimal fields that identify tuple

Primary key: Designated identifier for a tuple

Foreign key: Reference to another tuple

Referential Integrity

A database instance has *referential integrity* if all foreign key constraints are enforced no dangling references

Examples where referential integrity is not enforced

- HTML links
- Yellow page listing
- Restaurant menus
- Some relational databases!

How to Enforce Integrity Constraints

Run checks anytime database changes

On INSERT

what if new Enrolled tuple refers to non-existent student?
Reject insertion

On DELETE (many options)

what if Students tuple is deleted?

delete dependent Enrolled tuples

reject deletion

set Enrolled.sid to default value or null

(null means 'unknown' or 'inapplicable' in SQL)

General Constraints

Boolean constraint expression added to schema

Very powerful; not widely used

```
CREATE TABLE Enrolled(
    sid int,
    cid int,
    grade char(2),
    CHECK (
        grade = ‘A’ or grade = ‘B’ or
        grade = ‘C’ or grade = ‘D’ or
        grade = ‘F’
    )
)
```

Where do ICs come from?

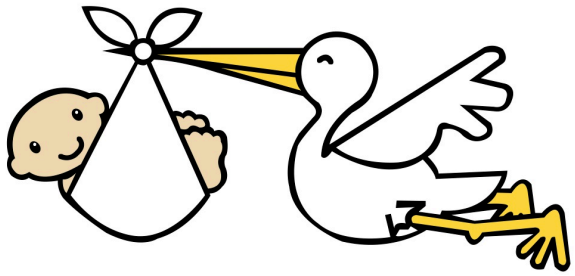
Based on application semantics and use cases

IC is statement about the world (all possible instances)

- Can't infer ICs by staring at an instance
- e.g. Is "login" a unique candidate key? Maybe?

Unique and foreign key ICs are very common

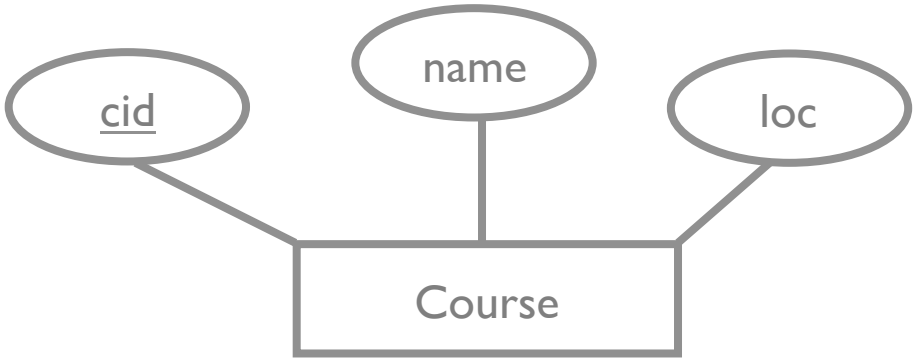
- (others less so)



Translating ER → Relational Models

Entity Set → Relation

Include all attributes
Entity set key become relation primary key

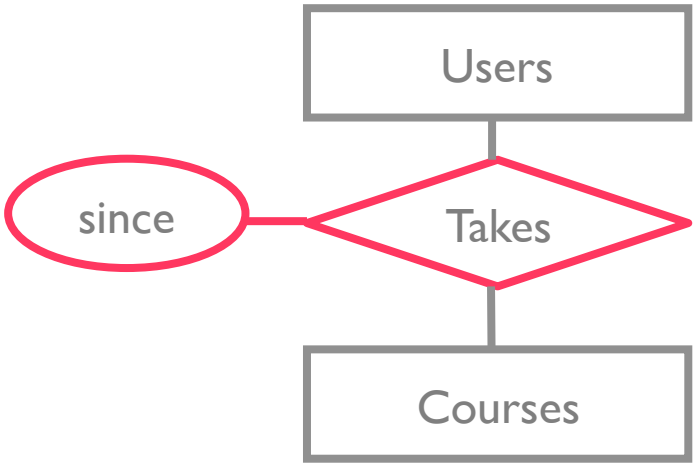



```
CREATE TABLE Course(
   cid int,
   name text,
   loc text,
   PRIMARY KEY (cid)
)
```

Relationship Set w/out constraint → Relation

Relation includes:

- All attributes of the relationship set
- Add keys for each entity set as foreign keys: *superkey*
  - (if there are other constraints: may not need all the keys)




```
CREATE TABLE Takes(
    uid int,
    cid int,
    since date,
    PRIMARY KEY (uid, cid),
    FOREIGN KEY (uid) REFERENCES Users,
    FOREIGN KEY (cid) REFERENCES Courses
)
```

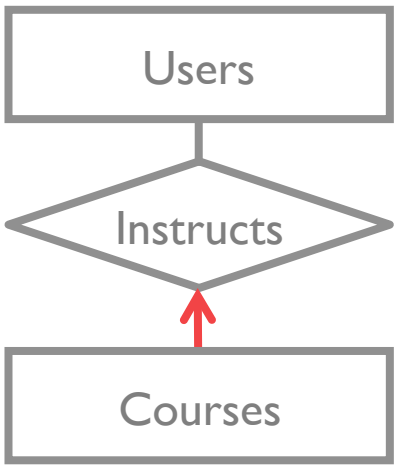
At Most One → Relation

Add relationship attributes (none here)

Add keys for entity set as foreign keys

What is the primary key?

Only cid: Zero or 1 Course in this relationship




```
CREATE TABLE Instructs(
    uid int,
    cid int,
    PRIMARY KEY (cid),
    FOREIGN KEY (uid) REFERENCES Users,
    FOREIGN KEY (cid) REFERENCES Courses
)
```

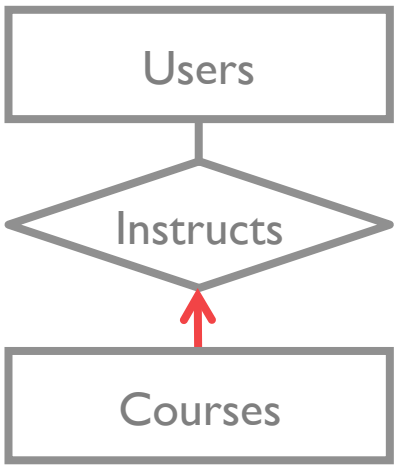
At Most One: Combine?

Zero or 1 courses, cid is primary key; Similar to ???

Combine Instructs attributes into Courses (preferred)

How to represent courses without instructor?

NULL value for uid (and other Instructs attributes)

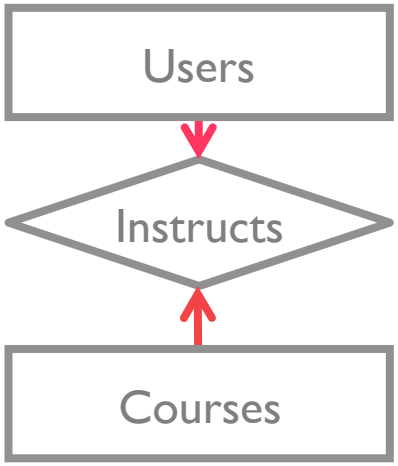



```
CREATE TABLE Course_Instructs(
    cid int
    uid int,
    name text,
    loc text,
    PRIMARY KEY (cid),
    FOREIGN KEY (uid) REFERENCES Users
)
```

Rename instructor_uid?

Key Constraint → Relation

- How to translate this ER diagram?
- NOTE: I'm pretty sure this is wrong! Hence hidden




```
CREATE TABLE Course_Instructs(
    cid int
    uid int,
    name text,
    loc text,
    username text,
    age text,
    PRIMARY KEY (cid),
    UNIQUE (uid)
)
```

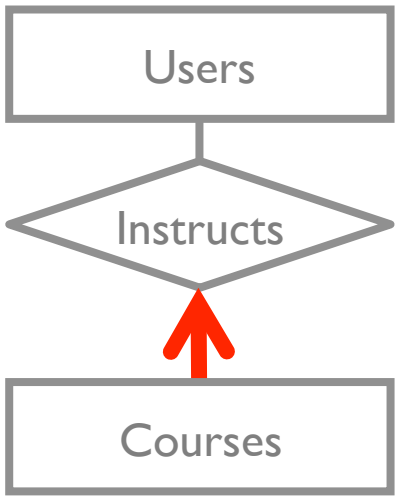
Exactly One Constraint → Relation

Represent Course must have Instructor?

Combine relationship into Courses; NOT NULL

What happens if we delete User who is Instructor?

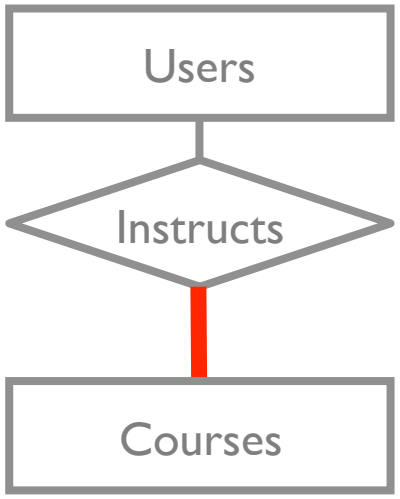
Default ON DELETE NO ACTION: Don't permit




```
CREATE TABLE Course_Instructs(
    cid int
    uid int,NOT NULL,
    name text,
    loc text,
    PRIMARY KEY (cid),
    FOREIGN KEY (uid) REFERENCES Users
)       ON DELETE NO ACTION
)
```

At Least One Constraint?

```
CREATE TABLE Instructs(
   cid int,
   uid int NOT NULL,
   PRIMARY KEY (cid, uid),
   FOREIGN KEY (cid) REFERENCES Course,
   FOREIGN KEY (uid) REFERENCES User)
```



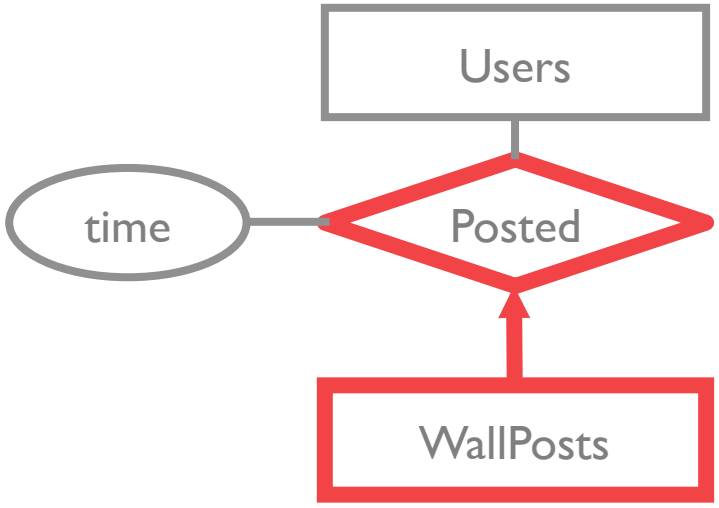
Same as no constraint!

Can’t be easily expressed

Weak Entity → Relation

Weak entity set and identifying relationship set are translated into a single table.

When the owner entity is deleted, all owned weak entities must also be deleted.




```
CREATE TABLE Wall_Posted(
    pid int
    post_text text,
    posted_time DATE,
    uid int,
    PRIMARY KEY (pid, uid),
    FOREIGN KEY (uid) REFERENCES Users
        ON DELETE CASCADE
)
```

ISA Hierarchies

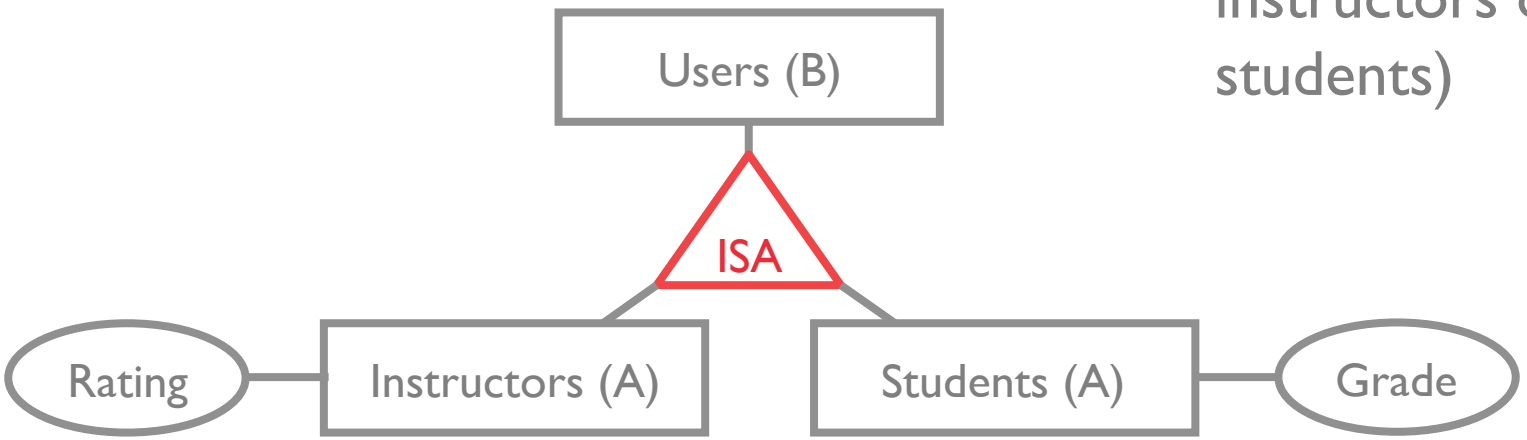
Option 1: Keep base relation

- Instructors & Students recorded in Users
- Extra info in Instructors or Students relation
- Primary key = key of parent table; is foreign key for child tables

Option 2: Only keep child relations

- Children copy attributes from Users

Only if covering constraint = yes (all users are instructors or students)



ISA Hierarchies

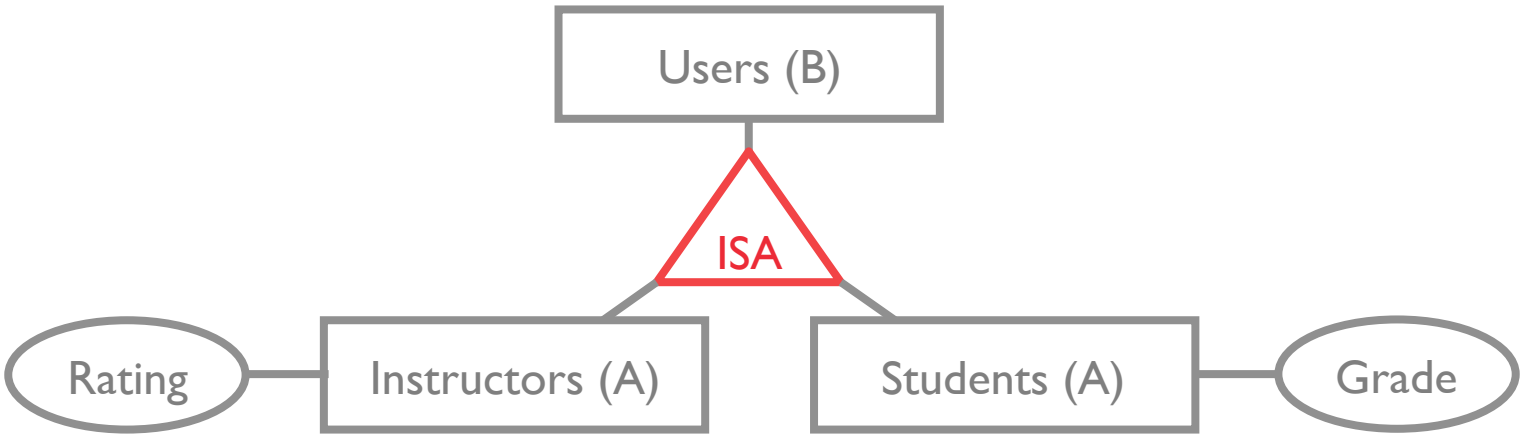
CREATE TABLE Users(uid int, name text, PRIMARY KEY(uid))

CREATE TABLE Instructors(uid int, rating int,

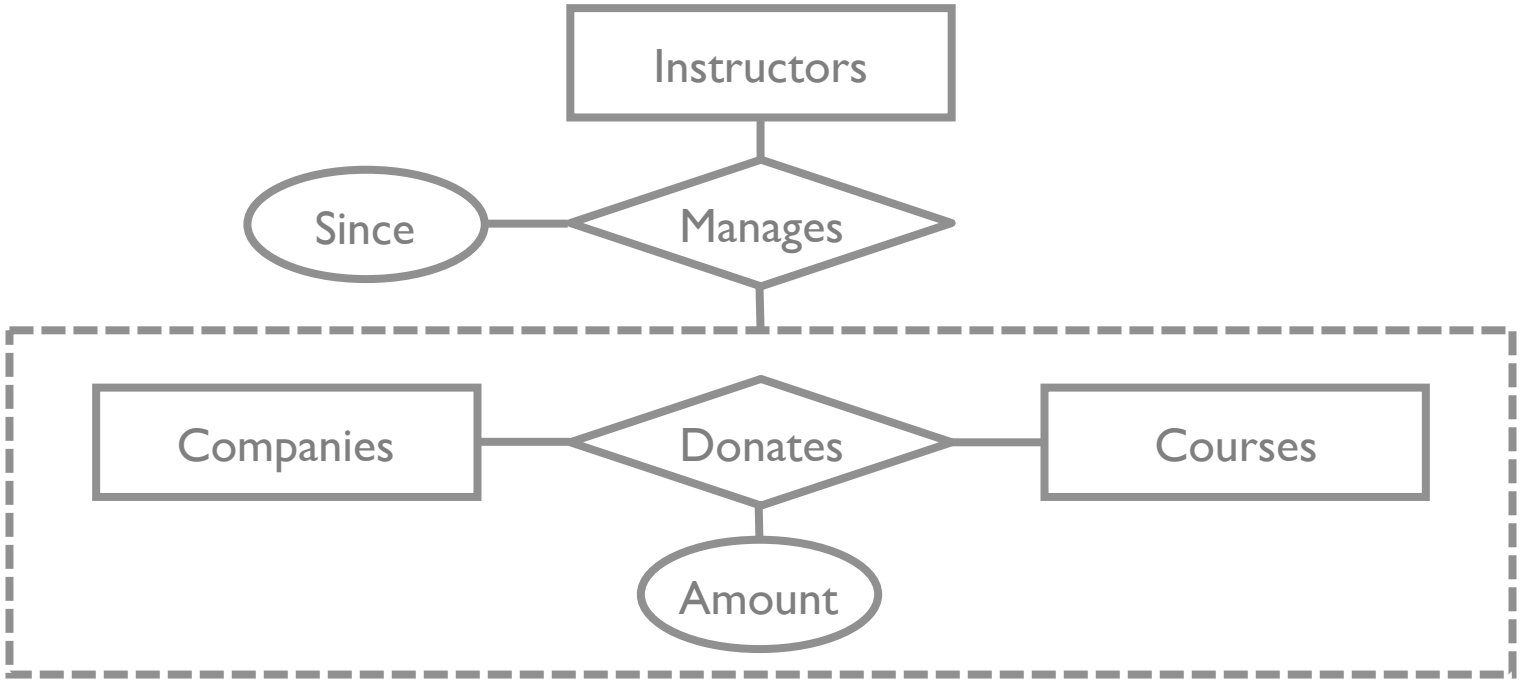
PRIMARY KEY(uid), FOREIGN KEY (uid) REFERENCES Users)

CREATE TABLE Students(uid int, grade char(2),

PRIMARY KEY(uid), FOREIGN KEY (uid) REFERENCES Users)

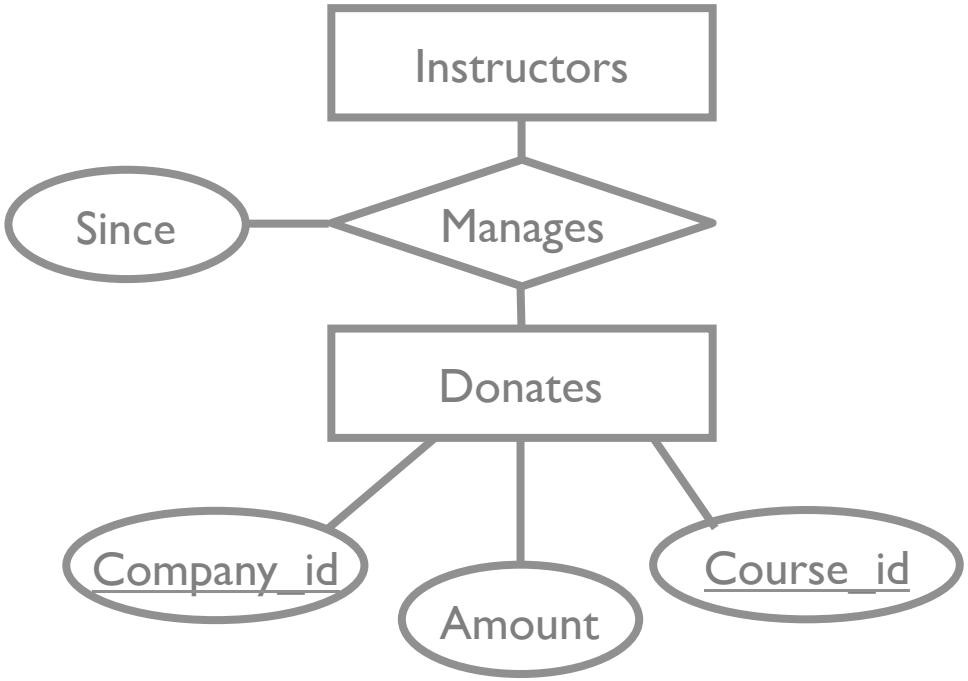


Aggregation



Aggregation

Convert the aggregated relationship into an entity
Convert like any other entity
E.g. Donates: PRIMARY KEY (Company_id, Course_id)
Manages: References this key



DML: Introduction to Queries

Query: Ask a question

Declarative: ask what you want, now how

Queries are high level, “readable”

- DBMS decides how to make it fast

Precise semantics for relational queries

- Lets DBMS choose different ways to run query while ensuring answer is the same

INSERT/DELETE

Add a tuple

```
INSERT INTO Students(sid, name, login, age, gpa)
VALUES (4, ‘wu’, ‘wu@cs’, 20, 5)
```

Delete tuples satisfying a predicate (condition)

```
DELETE FROM Students S
WHERE S.name = ‘wu’
```

Basic Single Table SELECT

SELECT *(output)* FROM *(input)*

WHERE *(condition)*

```
SELECT * FROM Students
SELECT name FROM Students
SELECT name FROM Students WHERE age < 21
SELECT name, login FROM Students WHERE gpa >= 3
```

sid	name	login	age	gpa
1	eugene	ewu@cs	20	2.5
2	luis	luis@cs	20	3.5
3	ken	ken@math	33	3.9

Ambiguous names

E.g. Students: (sid, name, …)

Enrolled: (sid, cid, grade)

Qualified names: Use table name: Students.age

Rename: AS (optional): shortcuts, ambiguity, clarity

SELECT S.sid, S.name FROM Students AS S

SELECT S.sid, S.name FROM Students S

Single Table Semantics

- A *conceptual evaluation method* for previous query:
  - 1. FROM clause: retrieve Students relation
  - 2. WHERE clause: Check conditions, discard tuples that fail
  - 3. SELECT clause: Delete unwanted fields

- Remember, this is *conceptual*. Actual evaluation will be *much* more efficient, but must produce the same answers.

Related data: Multiple tables

What does this return?

```
SELECT S.name, E.cid
FROM Students S, Enrolled E
WHERE S.sid = E.sid AND
      E.grade = 'A'
```

Enrolled

sid	cid	grade
1	2	A
1	3	B
2	2	A+

Students

sid	name
1	eugene
2	luis

Result

name	cid
eugene	2

Multi-Table Semantics

- Modify the FROM clause evaluation
  - 1. FROM clause: compute *cross-product* of Students and Enrolled

Enrolled

sid	cid	grade
1	2	A
1	3	B
2	2	A+

Students

sid	name
1	eugene
2	luis

Cross-product

sid	cid	grade	sid	name
1	2	A	1	eugene
1	3	B	1	eugene
2	2	A+	1	eugene
1	2	A	2	luis
1	3	B	2	luis
2	2	A+	2	luis

Multi-Table Semantics

- Modify the FROM clause evaluation
  - 1. FROM clause: compute *cross-product* of Students and Enrolled
  - 2. WHERE clause: Check conditions, discard tuples that fail
  - 3. SELECT clause: Delete unwanted fields