Universal Relations

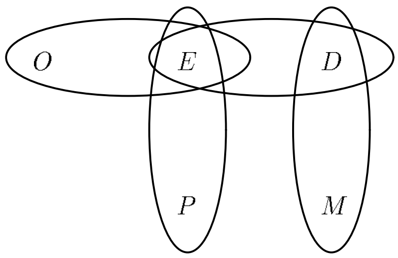
- The idea keeps getting reinvented about 3 times a year somewhere in the world.
  - So let's see it once and for all, and then if you ever need it you can just implement it rather than reinventing it.

Example of Problem that the UR Solves

Suppose we have relations ED, EO, EP, and DM, connecting employees to departments, phones, and offices, respectively, and departments to managers. To simplify SQL queries, we might define a view

```
create view EDOPM as
    select ED.E, ED.D, EO.O, EP.P, DM.M
    from ED, EO, EP, DM
    where ED.E = EO.E and EO.E = EP.E
        and DM.D = ED.D
```

Database Schema as a Hypergraph



- Consider a query "find the offices of employees managed by Sally." Without the view:

```
select EO.O
from EO, ED, DM
where EO.E = EM.E and DM.D = ED.D
    and M = 'sally'
```

- Using the view:

```
select O
from EDOPM
where M = 'sally'
```

- These two are not the same!
  - If, e.g., some employee reports to Sally but has no listed phone, then the query using the view fails to give that employee's office.

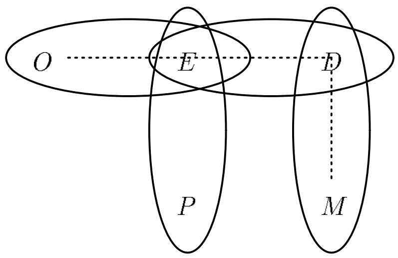
Simple UR Model

The problem above can be fixed if we take advantage of the fact that the various relation schemas use the same name for the same concept (and only for the same concept).

- We can thus imagine a *universal relation* whose attributes are all the attributes appearing in any schema.
  - The relation instance for this (imaginary) UR can be thought of as the natural join of all the relations, padded with nulls if necessary.
  - Sort of a multiway outerjoin.
  - But in queries, we never want to see the nulls.
- More precisely, to answer a query, we want to join only those relations that somehow connect the attributes of the query.
  - Thus, a dangling tuple in an irrelevant relation will not affect the answer.

Example

Query connecting O and M only goes through EO, ED, and DM, not EP.



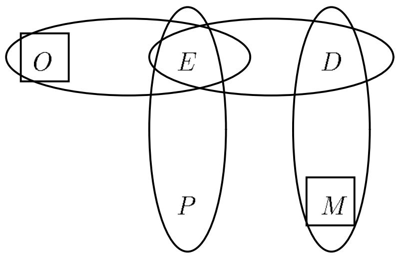
A UR Query-Answering Strategy

1. Construct the hypergraph for the schemas.
2. Mark the attributes (nodes) mentioned explicitly by the query as "sacred."
3. Do GYO reduction, but without deleting any sacred node.
4. The remaining hyperedges need to be joined and the query executed on that join.

Example

For our running example, O and M are sacred.

- Only ear EP can be removed.
- The remaining three hyperedges must be joined, so the query we answer is the same as the SQL query that doesn't use the $EDOPM$ view.



Problems With the Simple UR

The simple UR query-answering algorithm is OK if the hypergraph is acyclic and all apparent connections through the hypergraph are meaningful. But at least two problems surface:

1. If the hypergraph has cycles, there can be several paths connecting nodes, and the result is that these paths are "intersected" when we take the join.
   - I.e., to have a connection between two attributes, the data must support the connection along *all* paths.

---

2. In an acyclic hypergraph, to say that the join of hyperedges XY and YZ makes sense, we are asserting the multivalued dependency $Y \twoheadrightarrow X \mid Z$ holds.
   - If not, then we connect too many pairs of X-values and Z-values.

Example

If employees can have several offices and several phones, joining EO and EP to get office-phone connections is dubious.

- More likely each phone is on one office, and the hypergraph really should have a single EOP hyperedge.

Window Functions

In general, a *window function* for a universal relation maps the set X of attributes mentioned by the query to a relation $[X]$ over which the query is answered.

Example

The "simple" UR described above computes $[X]$ by:

1. Reduce the hypergraph using X as sacred nodes.
2. Take the natural join of the relations that remain.
3. Project this join onto X.

The Weak-Instance Window Function

A more conservative approach to establishing links is to assume that nothing is related unless it follows from a functional, multivalued, or other given dependency.

- Build a *weak instance* (of the universal relation) by taking the instances of the real relations and padding them with unique-value nulls.
    - ❖ We use $\perp_i$ for a null, where i is an integer chosen different from that of any other null.
- Then "chase" the weak instance by applying FD's to equate symbols (favor a real symbol over a null) and MVD's to generate new tuples.
- Finally, compute the *total projection* $[X]$ by projecting onto X only those tuples in the resulting weak instance that have no nulls in the columns for X.

Example

Assume relations ED and DM (employees-departments-managers) with functional dependencies $E \rightarrow D$ and $D \rightarrow M$. Suppose also that the current instances are: $ED = \{(e_1, d_1),\ (e_2, d_1),\ (e_3, d_2)\}$ and $DM = \{(d_1, m_1),\ (d_3, m_2)\}$.

- Let the query be "find all the employee-manager pairs," i.e., $X = \{E, M\}$.
- Then the weak instance starts out as

E	D	M
e_1	d_1	$\perp_1$
e_2	d_1	$\perp_2$
e_3	d_2	$\perp_3$
$\perp_4$	d_1	m_1
$\perp_5$	d_3	m_2

- The only equalities one can deduce are using $D \rightarrow M$, from which we can infer $\perp_1 = \perp_2 = m_1$.
- The weak instance becomes:

E	D	M
e_1	d_1	m_1
e_2	d_1	m_1
e_3	d_2	$\perp_3$
$\perp_4$	d_1	m_1
$\perp_5$	d_3	m_2

- The total projection onto EM is thus $\{(e_1, m_1),\ (e_2, m_1)\}$.