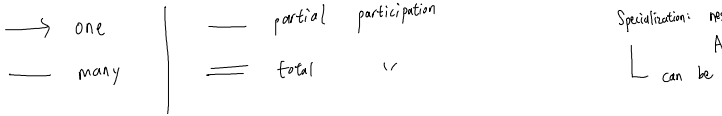
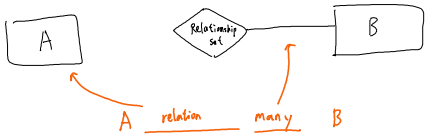
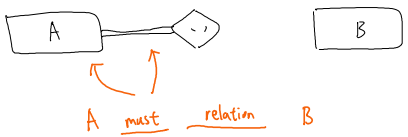


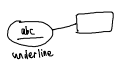
For mapping cardinality, A will look for the line after the relationship set (diamond)



For participation constraints, A will look for the line before



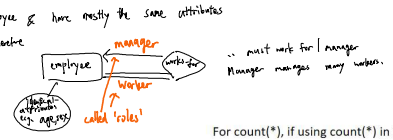
Keys:
- need to be unique
- Super key(s), e.g. combination of 2 keys {id, name}
- trim down
- Candidate key(s)
- pick one of them
- Primary key



Can we use keys from two entities to identify entity?
- Yes
- Use

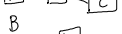
Must:
- Total participation: without it, can't identify
- Many-to-one: If one-to-one, discriminator becomes primary key
  If one-to-many, same theory
  If many-to-many, identifying entity set + weak entity set is not enough

Role: 'entity' rebranded X, not really ⇒ manager & worker are both employee & have mostly the same attributes



.. must work for 1 manager
Manager manages many workers.

Specialization: nested entities



A Is-A B

- can be total:

- partial:

disjoint or not disjoint.

- gender → single-valued
- phone num → multi-valued → 创建新的表储存，用 primary key 检索
- age → derived (from birth date)
  dashed

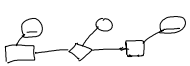
好多花样

Employee's phone num. 是作为 attribute 还是 entity?

Phone num. can be shared by multiple employees?

Relationship set as entity: 不同语义下使用

Relationship attribute



Many-to-Many
Many-to-one
One-to-one
} combine tables to cut duplicate info

When forming tables from specializations, should we duplicate root attributes into child? (e.g. address)
Storage redundancy v.s. Efficient retrieval of data?
- If specialization is total (particip), then no need to form root attributes table & move root attributes into child

How about non-disjoint cases?

Relationship set can become a table including entity sets' primary key as "foreign keys"

foreign keys: keys that actually refer to primary key in another entity set

referential constraint = foreign key MUST match existing values in the primary key column of referenced table

SELECT (name+5)/3 FROM stuff;

Executable instructions (included as column name)

(name+5)/3

```
For count(*), if using count(*) in SELECT statement,
You must apply "GROUP BY" to statements in SELECT that are not
aggregated (aka grouped), e.g. "GROUP BY A.name"
```

```
select A.name, count(*) from t3_authors A, t3_writes W
where A.authorID = W.authorID
group by A.name;
```

GROUP_CONCAT(FIELD SEPARATOR ', ') (not in lecture notes): Combine identical entries but with different value in FIELD together, which FIELD will concatenate together using separator specified in the function.

```
SELECT
   B.bookID,
   B.name,
   B.description,
   B.price,
   B.picture,
   GROUP_CONCAT(C.name SEPARATOR ', ') AS categories
FROM t3_books B, t3_categories C
WHERE C.name IN ("Computer Science", "Database")
GROUP BY B.bookID, B.name, B.description, B.price, B.picture;
```

Exercise 5: Subqueries
Question: Find books with prices higher than the average book price.

```
select * from t3_books B
where B.price > (select avg(B.price) from t3_books B);
```

nested query is needed since avg() is a "aggregated function", does not execute before "where" statement if we use "where B.price > avg(B.price);"

When comparing times:

```
SELECT r.reservation_ID, r.member_ID, m.name AS member_name,
   r.station_ID, r.reservation_datetime, r.collect_datetime, m.email
FROM Reservation r
   INNER JOIN Member m ON r.member_ID = m.member_ID
WHERE r.status = 1 -- Active reservation
   AND (strftime('%s', r.collect_datetime) - strftime('%s', r.reservation_datetime))
/ 60 <= 15
   -- we let strftime return the difference in seconds as %m will ignore the
seconds unit
   -- so we need /60 to convert it to minutes
ORDER BY r.reservation_datetime ASC, r.reservation_ID DESC;
SELECT strftime('%Y', date  column) FROM table_name;
```

COALESCE(something, default_value) provides a default value if inside is null/unknown.