CP363: Database I

by Scott King
Dr. Siu-Cheung Chau

Chatper 1: Introduction

1 Database Management Systems (DBMS)

A *DBMS* is a collection of software to enable a user to create, maintain and utilize a DB (Ex. Oracle, MySQL, etc).

1.1 Advantages of DBMS

- Efficient data access
- Data integrity and security
- Data administration
- Data independence
- Concurrent processing crash recovery
- Reduce application development time and cost

Concurrent Processing

{figure out diagram}

2 Logical Schema of a Database

- Schema is the data structureIBM had the hierarchical model, other companies created the Network and Lodeysol models
- Logical (conceptual) scheme is the logical data structure

3 Physical Schema

- How data is stored physically: tree, heap, etc
- View on the logical schema, different ways of viewing data; sometimes public facing

4 Logical Data Independence

- Changing to logical schema will not effect the view on it

5 Physical Data Independence

- Changing to physical schema will not effect the logical schema

5.1 Lingo

{figure out diagram}

6 Data Models

- IBM had the hierarchical model, other companies created the Network and Lodeysol models

Chapter 2: Introduction to the Relational model

1 Relational DB Model

- Data is stored using tables, also known as relations
- Each column has a name, AKA attribute
- Each row is record

1.1 Schema

- Defining the structure of a relation

Example 1.

```
student_table(Id int(8),
              Name varchar(8),
              Major varchar(8),
              GPA float);
```

- Tables are defined as such: $R(A_1, ..., A_n)$
- An *instance* is a record in a table

A **database instance** is all the data in the DB. **Database schema** contains schemas for all tables in the form $S = R_1, ...R_m$.

The table must satisfy four constraints:

1. Domain constraint
2. Key integrity constraint
3. Entity integrity constraint
4. Referential integrity constraint

Example 2.

Eid	Name	Dependent
1234	*Tom*	Mary
1234	*Tom*	Peter
2345	*John*	Tim
2345	*John*	Jane
3456	David	Jim

Remember **NO** duplicates. Instead change those records to:

Eid	Name	Dependent
1234	*Tom*	Mary,Peter
2345	*John*	Tim,Jane
3456	David	Jim

Each attribute can only contain *atomic* values. Ex. Cannot contain a set of values. FFS, do this:

Eid	Name	Dependent 1	Dependent 2
1234	*Tom*	Mary	Peter
2345	*John*	Tim	Jane
3456	David	Jim	--

This is also **NOT** ideal.

R_1:

Eid	Name
1234	*Tom*
2345	*John*
3456	David

R_2:

Eid	Dependent
1234	Mary
1234	Peter
2345	Tim
2345	Jane
3456	Jim

1. Key Integrity Constraint

- Observation: No tuples in a table are the same; in other words, every tuple in a table needs to be unique..SO DONT STORE RECORDS TWICE
- $\Rightarrow$ Each record, tuple, can be <u>uniquely identified</u> by a certain set of attributes
- The *super key* is a set of attributes that can uniquely identify a record (tuple)
- The trivial super key is the one with all the attributes
- A minimal super key is called a *candidate key*
- A candidate key be picked as the *primary key* $\rightarrow$ Refer to next Example 3
- Every relation must have a primary key `student_table(Id,Name,Major,GPA)` where `Id` is the primary key

Example 3. `id,dept_name` are super keys but not candidate keys. But two candidate keys are `id,Name` as they are unique. Then pick on candidate to be the primary, this `Id` will be picked as the primary key.

Example 4. {table}

We have the minimal super key = candidate key. $\{A, D\}$ is not a super key. $\{B, C\}$ is a super key. Is it a candidate key? No, thus, C is a super key $\Rightarrow C$ is a candidate key $\Rightarrow C$ is a primary key

2. Entity Integrity Constraint

- The primary key cannot be `null`; in our example, that means `Id` cannot be null
- The rest of the attribute can be `null`

3. Referential Integrity Constraint

- A *foreign key* must either contain a `null` value or a value in the **referenced key**

Example 5. Back to the example with R_1 and R_2. We have `R1(Eid,Name)` and `R2(Eid,Dependent)`.

Thus, `Eid`, is an foreign key in R_2 because `Eid` is the primary key in R_1. `Eid` in R_1 is the referenced key.

Chapter 7: DB Design

1. Requirement Analysis

- Data to be stored
- Applications
- Performance analysis

Example 6. Registration system

```
student records
class records
professor records
```

Actions:

- To register a student in the class
- Print class list
- Print professor schedule

2. Conceptual Database Design

- Entity-Relationship model (ER model)
- Entity is equivalent to record type
- The relationship between entities

3. Logical Database Design

- Convert the ER model into a relational model (ie. convert to tables - relations)

4. Scheme Refinement

- Theory of relational database - 1^{st}, 2^{nd}, 3^{rd}, 4^{th} BC (Boyce-Codd) normal forms

5. "Physical" Database Design

- Add index files to speed up certain applications

6. Application and Security Design

1 Entity-Relationship Model

Entity $\equiv$ record

- An entity is an object in the real word that is distinguishable from other objects

- A **diamond** indicates a relationship between two entities
- A **triangle** indicates a subclass of
- A **circle** represents an attribute
- A **rectangle** represents an entity

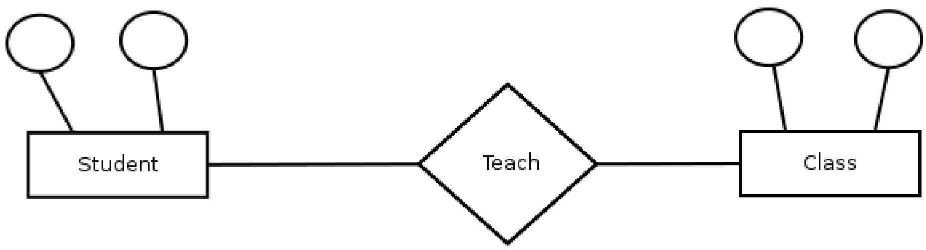
Example 7. A student record

These are attributes

`Id      Name      GPA      Major`

- An entity is a collection of entities of the same type
- All entities in the same set have the same attributes (cicles)
- *Student* is the entity and *{Id, Name, GPA, Major}* are the attributes (circles)
- Each entity has a primary key
- The key should depend on a real life situation and not the current set of data

2 Relationship Between Entities



2.1 Many to Many Relationship

1. A student can take many courses
2. A class can have many students

2.2 1 to Many Relationship

1. A professor can teach many classes
2. A class is taught by 1 professor

2.3 1 to 1 Relationship

1. A department has only one chair
2. A professor is the chair of only one department

Note: Every entity has a key and every relationship has a key.

A relationship does not have to be binary.

2.4 Total Participation

- Everyone is involved ⇒ total participation (**thick line**)

2.5 Recursive Relationship

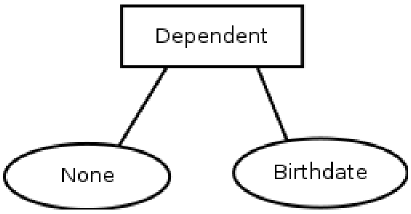
- An entity can only appear once in a design
- The key for the relationship (Id) - in our example

2.6 Weak Entity

- Entity without superkey

Note: Thick [box] lines implies weak entity.

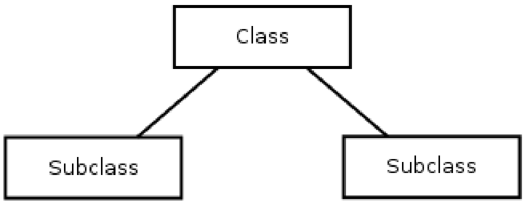
Example 8.

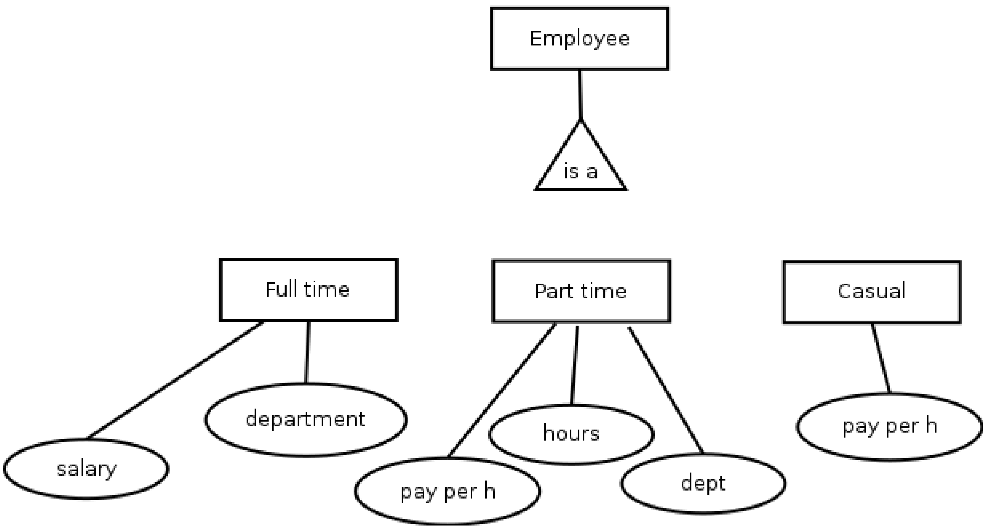


2.7 Strong Entity

- Entity with a super key (primary key)
- Everything so far is a strong entity

3 Class Hierarchies





The subclasses are disjoint.

4 Aggregation

- Abstraction to group relationships

Refer to customer - borrower - loan.

5 Issues in Conceptual Design Using the ER Model

1. Should a concept be modeled as an attribute or as an entity

   - Refer to employee one
   - Employees can have multiple addresses (sub address with employee to create new entity with attributes)
   - If we wanted to conduct searches in the address - it would be easier with address as an entity (ex. search for a city within a country)
   - Refer to picture - it is wrong, solution: look to next picture (make from and to as entity instead of attribute)

6 ER to Relational Database Mapping

Step 1

For each strong entity set, E, create a table (relation), R, that includes all attributes of E. The key for the entity will be the key for the table (relation).

$$\text{strong entity} \Rightarrow \text{table}$$

Step 2

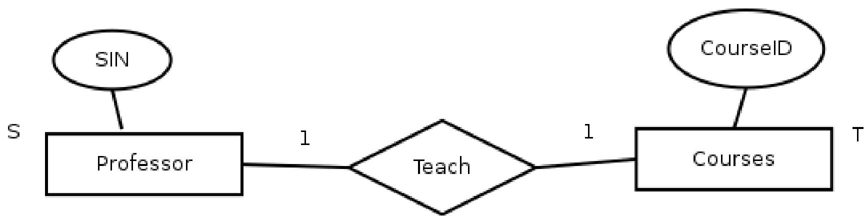
For each weak entity set, W, create a table, R, that includes all attributes of W and include it as a foreign key, the primary key of the owner entity. The key for the relation is the partial key and the key of the own entity.

{table for employee-dependents}

Step 3

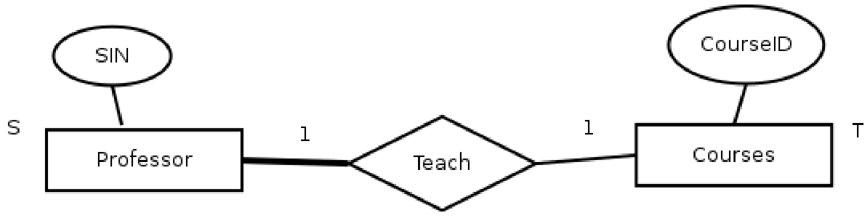
For each 1:1 relationship between two strong entitie, S and T. **No** new table (relation) is required. Just add the primary key of S as a foreign key in T.

Example 9.




```
professor(SIN, ..., ...)
courses(courseID, ..., ..., SIN)
```

If total participation exists in $\underline{S}$, we should add the primary key of T as a foreign key to $\underline{S}$.



Every professor much teach one course and the course cannot be taught by more than one professor.

Step 4

For each 1:n relationship between two strong entities S and T. No new table is required. Just add the primary key of S as a foreign key in T.

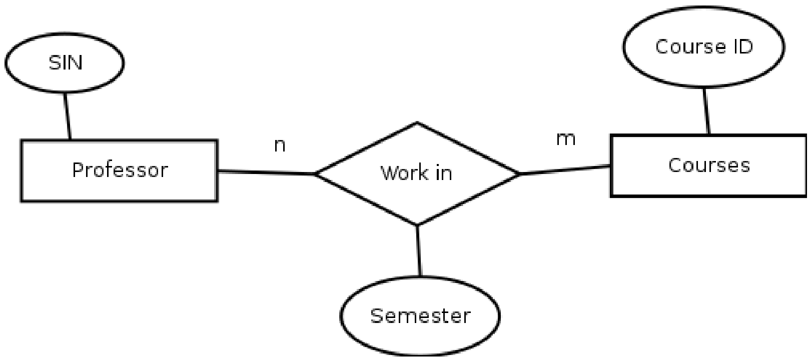
Example 10.

`Employee(id, ..., ..., departmentID)`

Step 5

For each many to many relatonship between S and T, create a new table (relation) that contains the primary key of S and T. The key for the new table is the primary key of both S and T together.

Example 11.

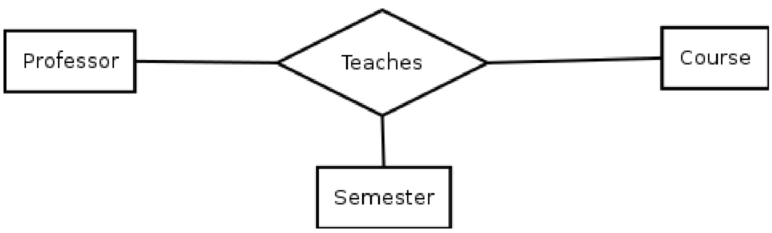


`Teaches(SIN, CourseID, semester)`

Step 6

n-ary relationships.

Example 12.

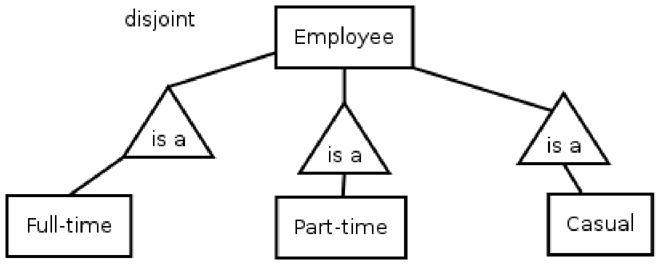


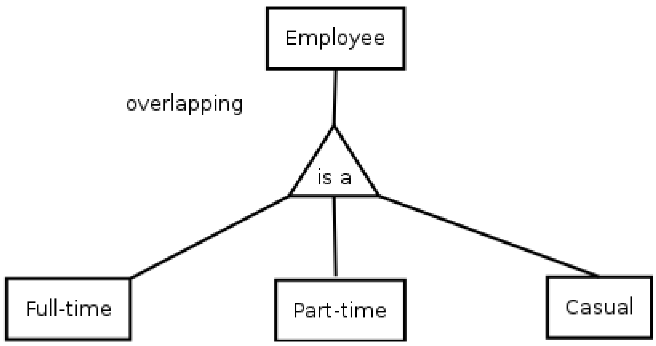
Teaches(SIN, semester_id, course_no)

Create a new table that contains the primary key of all the entities. The keys for the table are the primary keys of all entities together.

Step 7

Translating class hierarchies.





Disjoint implies that we create a table for each disjoint entity and it is necessary to create one for the parent. **Overlapping** implies that we create a table for all overlapping entities and create a table for the parent; only put the key of the parent in table of the subclass.

Step 8

Aggregates.

Create a table for an aggregate that contains the key for the aggregate and the key for the entity.

Example 13.

`Supervise(SIN,Graduate_SIN, p_no) all with dotted underlines`

7 Data Definition Language (DDL) - part of SQL

Example 14. Create a table in SQL

```
Employee(Eid,name,address,salary,Supervisor) - last one is dotted
Supervisor(Eid, S_ID) both dotted
```

Here's the code:

```
CREATE TABLE   Employee
```

```
(Eid           integer(9),      // or int(9)
 Name          varchar(30),
 address       varchar(50),
 salary        integer(5) unique,    // CANNOT BE null
 Sid           integer(9),
 primary key   (Eid)
 foreign key   (SupervisorID) reference (Employee))

CREATE TABLE   Supervisor
(Eid           int(9),
 Sid           int(9),
 primary key   (Eid,Sid),
 foreign key   (Eid) reference (Employee),
 foreign key   (Sid) reference (Employee))
```

7.1 Delete a Table

```
drop table Employee
```

7.2 Update a Table

Using `Employee` table, we want to add a new attribute.

```
Alter table Employee
    add status varchar(6)
```

Now we want to delete an attribute:

```
Alter table Employee
    drop name
```

We now want to delete a record with: `Employee id=1234`. `Employee` with `id=1234` may be a supervisor of other `Employee`'s. We need to update every other `Employee`'s `Supervisor`.

Chapter 7: Relation Algebra

1 Basic Operation

```
Select             æ (lowercase sigma)
Project            Π
Union              ∪
Set different      -
Cartesian product  ×
Rename             e
```

The first 3 +last are unary operations and the last three are binary operations.

Example 15. `Employee(Id,name,address,dept,salary)`

Temp $\leftarrow$ Π $\texttt{Employee}_{\texttt{Id;name}}$

Just choose a number of attributes from the table.

Select all records with a salary of ⩾80000

R $\leftarrow$ æ $\texttt{Employee}_{\texttt{salary}\geqslant 80000}$

Π `Employee`$_{\texttt{Id;none}}$ `(Select Employee`$_{\texttt{salary}\geqslant 80000}$`)`

```
æ          Id,name
From       Employee
Where      salary ⩾ 80000
```

Union is the same as set union. Set difference: only operands (relations) with the same attributes can be involved.

1.1 Cartesian Product

```
Employee(Id,name,dept,...)
Dept(dept_no,name,...)

Employee
1   1234    Tom    2   ...
2   2345    Mary   1   ...
3   3456    Jim    1   ...
4   4567    Kate   2   ...

Dept
a   1       Sales      ...
b   2       Accounting ...
```

Where this will produce a new table.

The universal relation contains all attributes. In order to avoid duplicates, we break it into many smaller relationships. We can use the cartesian product to join them back together. Unforunately, joining them back together doesn't always work 100%.

1.2 Join Product

- Natural join
- Equivalent join
- Conditional join

Natural join: $\ltimes\rtimes$. Example, `Employee` $\ltimes\rtimes$ `Department`

1. `Employee` $\times$ `Department`
2. $\sigma_{\texttt{Employee;Dnum}} = \texttt{Dnum}$

Equi-Join

$$\begin{aligned} \texttt{Employee}_{\texttt{dnum}} &\ltimes\rtimes \texttt{Department} \\ \equiv \sigma_{\texttt{Dnum}} \texttt{Employee} &\times \texttt{Department} \end{aligned}$$

Conditional Join

$$\begin{aligned} \texttt{Employee} &\ltimes\rtimes \texttt{Department} \\ E_1\,\texttt{salary} &> E_2\,\texttt{salary} \\ \equiv \sigma_{E_1\,\texttt{salary} > E_2\,\texttt{salary}} & \quad E_1 \times E_2 \end{aligned}$$

Example 16.

```
Sailors(Sid, Sname, age)
Boat(Bid, Bname, colour)
Reserves(Sid, Bid, Date)
```

<u>Query 1</u>

Find the name of the sailors who have reserved boat with `Bid=103`.

$$
\begin{aligned}
T_1 &\leftarrow \texttt{Sailor} \ltimes\rtimes \texttt{Reserves} \\
T_2 &\leftarrow \sigma_{\texttt{Bid=103}} T_1 \\
T_3 &\leftarrow \Pi_{\texttt{Sname}} T_2 \\
T_3 &\leftarrow \Pi_{\texttt{Sname}}(\texttt{Sailor} \ltimes\rtimes(\sigma_{\texttt{Bid=103}} \texttt{Reserves}))
\end{aligned}
$$

<u>Query 2</u>

Find the names of the sailors who have reserved at least one **red** boat.

$$
\begin{aligned}
T_1 &\leftarrow \texttt{Sailor} \ltimes\rtimes \texttt{Reserves} \ltimes\rtimes \texttt{Boat} \\
T_2 &\leftarrow \sigma_{\texttt{colour=red}} T_1 \\
T_3 &\leftarrow \Pi_{\texttt{Sname}} T_2 \\
T_1 &\leftarrow \Pi_{\texttt{Sname}}(\texttt{Sailor} \ltimes\rtimes \texttt{Reserves} \ltimes\rtimes (\sigma_{\texttt{colour=red}} \texttt{Boat}))
\end{aligned}
$$

<u>Query 3</u>

Find the names of the sailors who have reserved at least a **red** or a **green** boat.

Note: You can use red or green in one line, but it only works for *or*. You cannot do a similar operation with *and*. But it is preferred to split it up.

$$
\begin{aligned}
\text{Red} &\leftarrow \Pi_{\texttt{Sname}}(\texttt{Sailor} \ltimes\rtimes \texttt{Reserves} \ltimes\rtimes (\sigma_{\texttt{colour=red}} \texttt{Boat})) \\
\text{Green} &\leftarrow \Pi_{\texttt{Sname}}(\texttt{Sailor} \ltimes\rtimes \texttt{Reserves} \ltimes\rtimes (\sigma_{\texttt{colour=green}} \texttt{Boat})) \\
\text{Answer} &\leftarrow \text{Red} \cup \text{Green}
\end{aligned}
$$

<u>Query 4</u>

Find the names of the sailors who have reserved at least a **red** *and* a **green** boat.

$$
\text{Answer} \leftarrow \text{Red} \cap \text{Green}
$$

<u>Query 5</u>

Find the name of sailors who have reserved **all** red boats.

$$
\begin{aligned}
\text{All the red boats} &\leftarrow \sigma_{\texttt{colour=red}} \texttt{Boats} \\
T &\leftarrow \Pi_{\texttt{Sname}}(\texttt{Sailor} \ltimes \rtimes \texttt{Reserves} \,/\, \texttt{All red}) \\
&\text{or} \\
T_1 &\leftarrow \Pi_{\texttt{Sid;Bid}}(\texttt{Sailors} \times \texttt{All red}) \\
T_2 &\leftarrow T_1 - \Pi_{\texttt{Sid;Bid}}(\texttt{Reserves}) \\
T_3 &\leftarrow \Pi_{\texttt{Sid}}(\texttt{Sailors}) - \Pi_{\texttt{Sid}}(T_2)
\end{aligned}
$$

Division

Note: This is a not a basic operation, shame.

A	x	y
	α	1
	α	2
	α	3
	β	1
	γ	1
	θ	1
	θ	3
	θ	4
	ϵ	1
	ϵ	2
	κ	1
	κ	2
	κ	3

Where A and B are the same attribute.

B	y
	1
	2

$$\frac{A}{B} = \begin{array}{c} x \\ \alpha \\ \epsilon \\ \kappa \end{array}$$

First test: chapter 1 (1.1-1.5), chapter 2 (2.1-2.4), chapter 3 (3.2), chapter 7, chapter 6 (6.1).

Example 17. `Employee(id, name, dept, ..., salary)`

We need to find the total salary of all employees.

1. Retrieve all employee records
2. Write a program to sum all salaries

Denote $\mathcal{G}$ as an aggregate function (sum, average, max, min, count)

Thus, grab all salaries of employees: $\mathcal{G}_{\texttt{sum(salary)}}\texttt{Employee}$.

Find the number of records per `Employee`: $\mathcal{G}_{\texttt{count(*)}}\texttt{Employee}$

Example 18. `Employee(id, name, dept, ..., salary)`

We want to find the total salary for each department.

$$\texttt{dept}\mathcal{G}_{\texttt{sum(salary)}}\texttt{Employee}$$

Maximum salary for each dept:

$$\texttt{dept}\mathcal{G}_{\texttt{sum(salary)}}\texttt{Employee}$$

Find the average salary:

$$\texttt{dept}\mathcal{G}_{\texttt{avg(salary) as average}}\texttt{Employee}$$

Note: We created an alias for `avg`, that is `average.` Thus calling it will invoke `avg`.

Count the number of employees in each dept:

$$\texttt{dept}\mathcal{G}_{\texttt{count(id)}}\texttt{Employee}$$

Outer Join

There are 3 types:

1. Full outer join *bowtie with both top and bottom lines extended*
2. Left outer join *bowtie with left top and bottom lines extended
3. Right outer join *bowtie with right top and bottom lines extended*

Project	Π
Selection	σ

In SQL:

```
Select   <attribute1 > ... < attribute n >
  From   <relation1 > ... < relation n >
```

Example 19.

$$\Pi_{\mathsf{a_2,a_3}}(R_{1_{R_1 a_1 = R_2 a_2}} \ltimes \rtimes R_{2_{R_2 a_4 = R_3 a_4}} \ltimes \rtimes R_3)$$

Translation:

```
Select a2,a3
From   R1, R2, R3
where  R1a1 = R2a2 and R2a4 = R3a4
```

$$\mathcal{G}_{\texttt{sum(a}_1\texttt{)}} R_1$$

Translation:

```
Select sum(a1)
From   R1
```

Find the sum for each dept:

$$_{\text{dept}}\mathcal{G}_{\texttt{sum(salary)}} R_1$$

Translation:

```
Select sum(salary) as dept_total
From   R1
Group  by dept
```

You can also group would qualify having {some condition}.

1.3 Set Operations

- Union
- Intersect
- Minus/except

We **cannot** check if something is or isn't in a set (set membership test). We also **cannot** check if something exists or not (existence test).

Example 20. `Employee(Id, name, dept, dept_location, salary)`

```
Select    sum(salary) as dept_total
From      Employee
Group     by dept
```

$$\equiv {}_{\texttt{dept}}\mathcal{G}_{\texttt{sum(salary)}}\text{Employee}$$

Example 21. Given these three tables:

```
Sailors(Sid, name, rating, age)
Boats(Bid, name, colour)
Reserves(foreign: Sid, foreign: Bid, date)
```

Find the name of the sailor who has reserved boat with `Bid=103`.

```
Select     name
From       Sailors S, Reserved R
where      Bid=103 and S.sid=R.sid
```

Find the `Sid`'s of sailors who reserved a red boat.

```
Select     Sid
From       Boats B, Reserved R
where      B.bid=R.bid and colour='red'

Select     name
From       Sailor S, Boats B, Reserved R
where      S.sid=R.sid and B.bid=R.bid and colour='red'
```

Find the `colour`'s of boats reserved by Tom.

```
Select     code
From       Sailor S, Boats B, Reserved R
where      S.sid=R.sid and B.bid=R.bid and S.name='Tom'
```

Find the `color` of boats reserved by a sailor where `name` starts with a 'T'. We use something like this:

```
S.name     like 'T%'
```

where `%` is 0 to any arbitrary characters - a wildcard.

Find the `name` of the of sailors who have reserved a red boat but NOT a green boat.

```
SELECT     name
FROM       Sailors S, Boats B, Reserved R
WHERE      S.sid=R.bid and
           B.bid=R.bid and
           colour='red'

minus (or except)

SELECT     name
FROM       Sailors S, Boats B, Reserved R
WHERE      S.sid=R.bid and
           B.bid=R.bid and
           colour='green'
```

Find the `name` of a sailor who has reserved a boat with `bid=103`.

```
SELECT     name
FROM       Sailor S
WHERE      S.sid  IN
               SELECT     Sid
               FROM       Reserved
               WHERE      bid=103
```

Set Comparison Operators:

- Any
- ALL
- Some
- Every

To be used with $>$, $<$, $\leq$, $\geq$, $\neq$.

Example 22. Find the same of sailors whose rating is greater than some sailor called Tom.

```
SELECT     name
FROM       Sailor S
WHERE      S.rating > Any
                ( SELECT     S2.rating
                  FROM       Sailor S2
                  WHERE      S2.name = 'Tom' ) // also S2.name like 'T%'
```

Example 23. Find sailors who have reserved all the boats.

```
SELECT     B.bid
FROM       Boats B

SELECT     R.bid
FROM       Reserves R
WHERE      R.sid=S.sid
```

Example 24. Find the name of the oldest sailor:

- This is 2 steps:
  - Find the oldest sailor
  - Fnd the name of the oldest sailor

```
SELECT      name
FROM        Sailor

WHERE       age=(SELECT    max(age)
                 FROM      Sailor)
```

1.4 Null Values

- Aggregate functions → ignore null values
- Null ⇒ unknown

Three Value Logic:

- `True`, `False`, `unknown`
- `OR`
  - `T or unknown` ⇒ `True`
  - `False or unknown` ⇒ `unknown`
  - `unknown or unknown` ⇒ `unknown`
- `AND`
  - `T and unknown` ⇒ `unknown`
  - `False and unknown` ⇒ `False`
  - `unknown and unknown` ⇒ `unknown`
- `NOT`
  - `Not(unknown)` ⇒ `unknown`

Example 25. Full, left, right outer joins

Sailors

Sid	name	rating	age
22	Tom	7	22
31	John	10	15
58	Peter	7	40

Reserves

Sid	bid	date
22	30	10/10/15
31	40	10/11/15
32	50	12/11/15

Then:

```
Select     S.sid, R.bid
From       Sailors S {natural left outer join} Reserves R
```

The result:

```
22   30
31   40
22   50
58   null
```

1.5 Updating Tables

Example 26.

```
Update      Employee
Set         name='Tom'
Where       eid=1234
```

Options:

1. Restrict $\rightarrow$ default
2. Cascade
3. Set default
4. Set null

How to determine a good database design?

The main goal is this:

- **NO DUPLICATES!!!**

Problem with duplicates:

- Redundant storage
- Update, delete and insert anomalies

Functional Dependencies (FD)

Let R be a relation and let X, Y be non-empty sets of attributes in R.

FD $X \rightarrow Y$ is satisfied if for every tuple t_i and t_k in R if

$$t_i X = t_k X \Longrightarrow t_i Y = t_k Y$$

Armstrong's Axiom

Reflexivity

$$\begin{array}{c} A \rightarrow A \\ A B \rightarrow A \quad \text{or} \quad A B \rightarrow B \end{array}$$

Augmentation

$$\begin{array}{rcl} A & \rightarrow & B \\ x A & \rightarrow & x B \end{array}$$

<u>Transitive</u>

$$\begin{array}{rcl} A & \rightarrow & B \\ B & \rightarrow & C \\ A & \rightarrow & C \end{array}$$

<u>Union</u>

$$\begin{array}{rcl} X \rightarrow Y & , & X \rightarrow Z \\ X & \rightarrow & YZ \end{array}$$

<u>Decomposition</u>

$$\begin{array}{rcl} X & \rightarrow & YZ \\ X \rightarrow Y & , & X \rightarrow Z \end{array}$$

Closures of FD's

- Denoted by FD^+
- The closure is all FD that can be generated
- Ex. $R = (A, B, C)$, where $F = \{A \rightarrow B, B \rightarrow C\} \longrightarrow$ Find F^+

Let F be the set of FD's. The closure of F is denoted as F^+.

Note: All FD's implied by F is F^+.

Example 27. If $R(A, B, C, D)$, we can break it up into $R(A, B, C)$ and $R(C, D)$, we can break it up into

$$\begin{array}{rl} F_1 & = \\ F_1^+ & = \\ F_2 & = \\ F_2^+ & = \end{array}$$

Where $F_1^+ \equiv F_2^+$. The decomposition **preserves** all FD.

Attribute Closure

Example 28. $F = \{A \rightarrow B, B \rightarrow C\}$. Find the attribute closure of A, B and C.

$$\begin{array}{rcl} A^+ & = & A \\ & = & AB \rightsquigarrow \text{because } A \rightarrow B \\ & = & ABC \rightsquigarrow \text{because } B \rightarrow C \\ B^+ & = & B \\ & = & BC \rightsquigarrow \text{because } B \rightarrow C \\ C^+ & = & C \end{array}$$

From the attribute closure, we can determine the key. The key is A because $A^+ = R$.

To check if a given FD $B \rightarrow A$ is in F^+, check if A is in B^+.

Example 29. $R = (A, B, C, G, H, I)$ with $F = \{A \to B, A \to C, CG \to H, CG \to I\}$. Find a key in R (primary).

$$\begin{aligned}
A^+ &= A \\
&= AB \rightsquigarrow \because A \to B \\
&= ABH \rightsquigarrow \because A \to H \\
&= ABCH \rightsquigarrow \because A \to C \\
G^+ &= G \\
I^+ &= I \\
AG^+ &= AG \\
&= ABG \\
&= ABGH \\
&= ABCGH \\
&= ABCGHI \\
\therefore AG^+ &= R
\end{aligned}$$

Normal Forms

1$^{\text{st}}$ Normal Form → 1NF

- Every attribute contains only atomic values

2$^{\text{nd}}$ Normal Forms → 2NF

- In 1NF and every nm-key attribute must be fully functionally dependent of the entire key

Boyce-Codd Normal Form → BCNF

- R is in BCNF if and only if for every $X \to A$ in F, where $X \to F$ is a trivial FD *or* X is a superkey of R

Example 30. If $R(A, B, C)$ and $F = \{A \to B, B \to C\}$.

Is R in BCNF? $A \to B$. Is A a superkey of R? Yes, because $A \to B$ and $B \to C \Rightarrow A \to C$.

$B \to C$

$$\begin{aligned}
B^+ &= B \\
&= BC \rightsquigarrow \because B \to C \\
B^+ &\neq R
\end{aligned}$$

This is because B is not a superkey, $\therefore$ R is not in BCNF.

Example 31. Based on a DB of some people:

$$\begin{aligned}
\text{FD} &= \text{id} \to \text{name, rank, hours worked} \\
&\quad\ \text{rank} \to \text{hourly wage}
\end{aligned}$$

We have $E(\text{id, name, rank, hours worked, hourly wage})$ and then $R_1(\text{id, name, rank, hours worked})$ and $R_2(\text{rank, hourly wage})$.

BCNF has **NO** data redundancy.

Example 32. Decompose a relation into relations in BCNF. $R = (C, S, J, D, P, Q, V)$. C is the key. $SD \rightarrow P$, $J \rightarrow S$.

R is in 2NF, but not BCNF.

1 Decomposing a Schema into BCNF

- Suppose we have a schema R, and a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF, we decompose R into
    - $(\alpha \cup \beta)$
    - $(R - (\beta - \alpha))$