

關係論理

論理 (Logic, Calculus) とは

数理論理学
(Mathematical Logic)

- 論理学を数学的に形式化
- 論理式と推測規則

論理計算
(Calculus)

- 計算的な側面を強調
- λ 計算 (λ calculus)

命題論理(Propositional Logic/calculus)

論理式

- 個々の命題変数(P, Q など)は論理式
- P, Q が論理式なら, $\neg P$, $P \wedge Q$, $P \vee Q$, $P \rightarrow Q$ は論理式

推論規則

- 三段論法(modus ponens)
- $P \wedge P \rightarrow Q$ ならば Q

述語論理(predicate logic/calculus)

論理式の要素

- 個体変数 : x, y, z など
- 述語記号 : P, Q, R など

論理式の定義

- 原子式 : n 項の述語記号 P と 変数 x_1, \dots, x_n について $P(x_1, \dots, x_n)$
- 合成式 : A, B が 論理式 ならば,
 $\neg A, A \wedge B, A \vee B, A \rightarrow B, \forall x A, \exists x A$
は 論理式
- 定数と関数記号も体系に含める.

推論規則

- NK, LK などがある.

述語論理(predicate logic/calculus)

論理式の例: 大学生は必ず授業科目を履修する

- $P(x)$: x は大学生である
- $Q(y)$: y は授業科目である
- $R(x,y)$: x は y を履修する.
- $\forall x (P(x) \rightarrow \exists y (Q(y) \wedge R(x,y)))$

変数の出現(occurrence): 自由(free), 束縛(bound)

- $\exists y (Q(y) \wedge R(x,y))$ という論理式
 - x の出現は自由; y は $\exists y$ に束縛

関係論理 (Relational Calculus)

一階述語論理を基礎としたデータ操作体系

論理式を用いて対象となる関係を宣言的に記述

2種類の関係論理

- タプル関係論理: *Tuple relational calculus* (TRC)
 - 関係の組に着目
- ドメイン関係論理: *Domain relational calculus* (DRC)
 - 関係内の個々の値に着目

一階述語論理との違い

- 関係の構造を想定
- 関数記号を使わない
- 推論規則がない

関係代数の基本演算をすべて表現可能; より宣言的

例

Sailors

<u>sid</u>	sname	rating	age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Reserves

<u>sid</u>	<u>bid</u>	day
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Boats

<u>bid</u>	bname	color
101	Interlake	Blue
102	Interlake	Red
103	Clipper	Green
104	Marine	Red

タプル関係論理

- タプル変数を利用する
- 例：

(1) Ratingが7以上のsailorsの名前と年齢を示せ

$$\{P \mid \exists S \in \text{Sailors} (S.\text{rating} > 7 \wedge P.\text{name} = S.\text{sname} \wedge P.\text{age} = S.\text{age})\}$$

$$\pi_{\text{sname}, \text{age}}(\sigma_{\text{rating} > 7} \text{Sailors})$$

(2) boat 103を予約したsailorsの名前を示せ

$$\{P \mid \exists S \in \text{Sailors} \exists R \in \text{Reserves} (R.\text{sid} = S.\text{sid} \wedge R.\text{bid} = 103 \wedge P.\text{name} = S.\text{sname})\}$$

$$\pi_{\text{sname}}(\sigma_{\text{bid} = 103} (\text{Reserves} \bowtie \text{Sailors}))$$

タプル関係論理

- 例:

(3) 赤いboatを予約したsailorsの名前は？

$$\{P \mid \exists S \in \text{Sailors} \quad \exists R \in \text{Reserves} \quad \exists B \in \text{Boats} \quad (R.sid = S.sid \wedge R.bid = B.bid \wedge B.color = 'red' \wedge P.name = S.sname)\}$$
$$\{P \mid \exists S \in \text{Sailors} \quad \exists R \in \text{Reserves} \quad (R.sid = S.sid \wedge P.name = S.sname \wedge \exists B \in \text{Boats} \quad (R.bid = B.bid \wedge B.color = 'red'))\}$$
$$\pi_{sname}((\sigma_{color='red'} Boats) \bowtie \text{Reserves} \bowtie \text{Sailors})$$

タプル関係論理

- 例:

(4) すべてのboatを予約したsailorsの名前は？

$\{P \mid \exists S \in \text{Sailors} \quad \forall B \in \text{Boats} \quad (\exists R \in \text{Reserves} \quad (R.sid = S.sid \wedge R.bid = B.bid \wedge P.name = S.sname))\}$

$\rho (Tempсид, (\pi_{sid,bid} \text{Reserves}) / (\pi_{bid} \text{Boats}))$

$\pi_{sname} (Tempсид \bowtie \text{Sailors})$

ドメイン関係論理

属性変数を利用

- ❖ 問合せ(Query):

$$\{ \langle x_1, x_2, \dots, x_n \rangle \mid p(\langle x_1, x_2, \dots, x_n \rangle) \}$$

- ❖ 結果 : $p(\langle x_1, x_2, \dots, x_n \rangle)$ が真 (true) となるすべてのタプル $\langle x_1, x_2, \dots, x_n \rangle$

- ❖ 論理式は再帰的に定義

- ❖ 原子論理式 (タプルを抽出する論理式,

- ❖ 値を比較する論理式)

- ❖ 論理結合子 (\wedge , \vee , \neg , \forall , \exists) で

- ❖ 論理式を結合したものも論理式である.

ドメイン関係論理式

❖ 原子論理式:

- $\langle x_1, x_2, \dots, x_n \rangle \in Rname$
- $X op Y$, or $X op$ constant
- op is one of $<, >, =, \leq, \geq, \neq$

❖ 論理式:

- 原子論理式, または,
- $\neg p, p \wedge q, p \vee q$, と q は論理式. または,
- $\exists X(p(X))$, X が $p(X)$ 内で自由. または,
- $\forall X(p(X))$, X が $p(X)$ 内で自由
- 存在記号 \exists と \forall は X を束縛(bind)する.
- 束縛されていない変数は自由である.

自由変数と束縛変数(Free and Bound Variables)

- ❖ 問合せ (Query)

$$\{ \langle x_1, x_2, \dots, x_n \rangle \mid p(\langle x_1, x_2, \dots, x_n \rangle) \}$$

- ❖ 制約: x_1, \dots, x_n は論理式 p 内の唯一の自由変数である.

例題：

rating が7超えたsailorsは？

$$\left\{ \langle I, N, T, A \rangle \mid \langle I, N, T, A \rangle \in \textit{Sailors} \wedge T > 7 \right\}$$

- ❖ 条件 $\langle I, N, T, A \rangle \in \textit{Sailors}$ は、ドメイン変数 I, N, T and A が関係 *Sailors* の属性に対応させる。
- ❖ `|' (such that) 左辺の $\langle I, N, T, A \rangle$ は、 $T > 7$ を満たす各タプル $\langle I, N, T, A \rangle$ は答えに含むことを示す。

$$\{P \mid \exists S \in \textit{Sailors} (S.\text{rating} > 7 \wedge P.\text{sid} = S.\text{sid} \wedge P.\text{sname} = S.\text{sname} \wedge P.\text{rating} = S.\text{rating} \wedge P.\text{age} = S.\text{age})\}$$

rating が7超えたsailorsの名前は？

$$\{N \mid \exists I \exists T \exists A (\langle I, N, T, A \rangle \in \text{Sailors} \wedge T > 7)\}$$

$$\{P \mid \exists S \in \text{Sailors} (S.\text{rating} > 7 \wedge P.\text{sname} = S.\text{sname})\}$$

boat 103 を予約したratingが7超えたsailors を示せ

$$\left\{ \langle I, N, T, A \rangle \mid \langle I, N, T, A \rangle \in \text{Sailors} \wedge T > 7 \wedge \exists Ir, Br, D \left(\langle Ir, Br, D \rangle \in \text{Reserves} \wedge Ir = I \wedge Br = 103 \right) \right\}$$

- ❖ $\exists Ir, Br, D \ (\dots) : \ \exists Ir \left(\exists Br \left(\exists D(\dots) \right) \right)$

- ❖ ヨを用いて、関係Reservesから条件を満たすタプルを見つけてSailorsと結合する。

Find sailors rated > 7 who've reserved a red boat

$$\left\{ \langle I, N, T, A \rangle \mid \langle I, N, T, A \rangle \in \text{Sailors} \wedge T > 7 \wedge \right. \\ \exists \langle Ir, Br, D \rangle \left(\langle Ir, Br, D \rangle \in \text{Reserves} \wedge Ir = I \wedge \right. \\ \left. \left. \exists \langle B, BN, C \rangle \left(\langle B, BN, C \rangle \in \text{Boats} \wedge B = Br \wedge C = 'red' \right) \right) \right\}$$

❖ 括弧の範囲に注意

Find sailors who've reserved all boats

$$\left\{ \langle I, N, T, A \rangle \mid \langle I, N, T, A \rangle \in \text{Sailors} \wedge \right.$$
$$\forall B, BN, C \left(\neg \langle B, BN, C \rangle \in \text{Boats} \right) \vee$$
$$\left. \left(\exists Ir, Br, D \left(\langle Ir, Br, D \rangle \in \text{Reserves} \wedge I = Ir \wedge Br = B \right) \right) \right\}$$

Find sailors who've reserved all boats

❖ シンプル・直観的！

$$\begin{aligned} & \left\{ \langle I, N, T, A \rangle \mid \langle I, N, T, A \rangle \in Sailors \wedge \right. \\ & \forall \langle B, BN, C \rangle \in Boats \\ & \left. \left(\exists \langle Ir, Br, D \rangle \in Reserves(I=Ir \wedge Br=B) \right) \right\} \end{aligned}$$

$$\rho (Tempsids, (\pi_{sid,bid} Reserves) / (\pi_{bid} Boats))$$

$$\pi_{sname} (Tempsids \bowtie Sailors)$$

Find sailors who've reserved all red boats

$$\left\{ \langle I, N, T, A \rangle \mid \langle I, N, T, A \rangle \in \text{Sailors} \wedge \right. \\ \forall \langle B, BN, C \rangle \in \text{Boats} \\ \left. \left(C \neq 'red' \vee \exists \langle Ir, Br, D \rangle \in \text{Reserves} (I = Ir \wedge Br = B) \right) \right\}$$

$$C = 'red' \rightarrow \exists \langle Ir, Br, D \rangle \in \text{Reservers} (I = Ir \wedge Br = B)$$

安全でない問合せ(Unsafe Queries)

文法的に正しいが、答えが無限集合になる問合せは安全でない(unsafe)という。

- e.g.,

$$\{S \mid \neg(S \in Sailors)\}$$

関係代数で表現できる問合せ

<-> DRCまたはTRCで安全な問合せとして表現できる。

表現能力

関係完備(*Relational Completeness*): Query 関係論理/関係代数で表現できる問合せを表現できること。

SQL, 関係代数, 関係論理(Safe)の記述操作能力が等しい.

まとめ

関係論理:より宣言的

関係代数:より操作的・手続き的

関係代数・安全の関係論理の記述操作能力が等しい;関係完備

関係代数

関係の問合せ言語

- 問合せ言語: データベースにあるデータの操作と検索を行う.
- 関係モデルは、シンプルかつパワフルな問合せ言語をサポート:
 - 理論基盤
 - 最適化可能
- 問合せ言語 != プログラミング言語!
 - 問合せ言語は、チューニング完全である必要がない
 - 複雑な計算に用いない
 - 大きいデータセットへの容易かつ効率よいアクセスをサポート

二つの操作体系

- 二つの数学的問合せ言語体系
- SQLなどの実際の言語の基礎
 - 関係代数(*Relational Algebra*): 操作を表現, 手続き的にデータを扱う.
操作の実行プランの表現によく利用される.
 - 関係論理(*Relational Calculus*): 望む結果を表現, 宣言的にデータを扱う.

Preliminaries

問合せ先と問合せ結果は関係のインスタンスである。

- 入力である関係のスキーマは固定
- 結果のスキーマも固定！

フィールドの記述方法

- フィールドのポジションで指定・記述
- フィールドの名前で指定・記述
- SQLでは両方サポート

利用する関係インスタンス

R1

<u>sid</u>	<u>bid</u>	<u>day</u>
22	101	10/10/96
58	103	11/12/96

s1

<u>sid</u>	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

s2

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

- “Sailors” と “Reserves” の二つの関係
- ポジションと名前記法の両方を利用
- 問合せ結果におけるフィールドの名前は、問合せ先のフィールドの名前を継承とする。

代数 (Algebra) とは

演算対象の集合 + 演算子 (operator) の集合で構成される数学的体系

例

- ブール代数 : {0,1} + 論理演算 (AND, OR, NOT等)
- 線形代数 : 行列 + 行列に対する演算

関係代数

- 対象 : 関係
- 演算子 : 射影、選択など。一つまたは複数の関係を引数とし、処理を行い、結果となる関係を返す
- 演算の組合せが可能

関係代数

- 基本的な演算子:
 - Selection (σ) 関係の選択(行の抽出)
 - Projection (π) 関係の射影(カラムを抽出)
 - Cross-product (\times) 二つの関係の直積
 - Set-difference (−) 二つの関係の差
 - Union (\cup) 二つの関係の和
- その他の演算子:
 - Intersection(共通部分集合), join (結合), division(商), renaming(改名)

Projection(射影)

- 関係が持つ属性のうち, 指定した属性だけを残し他を削除する
- 結果のスキーマは, 射影リストに列挙されているフィールドから構成される.
- 射影演算は重複(duplicates)を除去
 - 注: 実際のシステムでは, ユーザが明示しなければ, 除去しない.

s_2	sid	sname	rating	age
	28	yuppy	9	35.0
	31	lubber	8	55.5
	44	guppy	5	35.0
	58	rusty	10	35.0

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

$\pi_{sname, rating}^{(S2)}$

age
35.0
55.5

$\pi_{age}^{(S2)}$

Selection(選択)

- 関係のタプル(行)のうち、指定した選択条件(*selection condition*)を満たすものだけを残す
- 出力となる関係のスキーマは入力と同じ
- 選択条件:
 - 属性と定数の比較
 - 属性と属性の比較
 - 上記の組合せ

s_2	sid	sname	rating	age
	28	yuppy	9	35.0
	31	lubber	8	55.5
	44	guppy	5	35.0
	58	rusty	10	35.0

sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

$$\sigma_{rating > 8}^{(S2)}$$

sname	rating
yuppy	9
rusty	10

$$\pi_{sname, rating}(\sigma_{rating > 8}^{(S2)})$$

Union(和)

- 二つの関係の和集合をとる。
- 制約: **和両立**(union compatibility)
 - 二つの関係の次数は同じ
 - 属性のドメインも同じ
- 和集合の結果の関係が持つ属性名は左辺の関係と同一

<i>s1</i>	<u>sid</u>	sname	rating	age
	22	dustin	7	45.0
	31	lubber	8	55.5
	58	rusty	10	35.0

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

$$S1 \cup S2$$

<i>s2</i>	<u>sid</u>	sname	rating	age
	28	yuppy	9	35.0
	31	lubber	8	55.5
	44	guppy	5	35.0
	58	rusty	10	35.0

Difference(差)

- ・ 差集合をとる
- ・ 制約: 和両立
- ・ 差集合の結果の関係が持つ属性名は左辺の関係と同一

sid	sname	rating	age
22	dustin	7	45.0

$S1 - S2$

s_1	sid	sname	rating	age
	22	dustin	7	45.0
	31	lubber	8	55.5
	58	rusty	10	35.0

s_2	sid	sname	rating	age
	28	yuppy	9	35.0
	31	lubber	8	55.5
	44	guppy	5	35.0
	58	rusty	10	35.0

Intersection(共通部分集合)

- 二つの関係の交わりをとる
- 制約: 和両立
- 差演算で表現可能

sid	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

$$S1 \cap S2$$

s1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

s2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Cross-Product(直積)

- 左辺の関係のそれぞれのタプルを、右辺の関係のそれぞれのタプルと連結
- 演算結果の関係の属性名
 - 二つの関係の属性名に同じものがなければそのまま引き継ぐ
 - 同じものがあれば、 $S1.X$, $R1.X$ などとして区別

$R1$

<u>sid</u>	<u>bid</u>	<u>day</u>
22	101	10/10/96
58	103	11/12/96

$S1$

<u>sid</u>	<u>sname</u>	<u>rating</u>	<u>age</u>
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

(<u>sid</u>)	<u>sname</u>	<u>rating</u>	<u>age</u>	(<u>sid</u>)	<u>bid</u>	<u>day</u>
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

Renaming(改名)

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

$$\rho(C(1 \rightarrow sid1, 5 \rightarrow sid2), S1 \times R1)$$

Joins(結合)

直積と選択の組合せ

- 条件結合(Condition Join): $R \bowtie_C S = \sigma_C(R \times S)$

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	58	103	11/12/96

$$S1 \bowtie_{S1.sid < R1.sid} R1$$

- 結果(関係)のスキーマは、直積と同様。
- タプル数が直積より少ない(Why?)
- theta-joinとも呼ぶ。

Joins(結合)

- 等結合 (Equi-Join): 比較演算子が“=”である場合の条件結合

sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
58	rusty	10	35.0	103	11/12/96

$$S1 \bowtie_{sid} R1$$

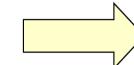
- 自然結合 (Natural Join): すべての同じ属性名の属性について等結合を行う.

Joins(結合)

- 自然結合(Natural Join):
すべての同じ属性名の属性について等結合を行う.
- 二つの関係の等しい属性名の等しい値どうしを結びつけることによって, もとの二つの関係の属性集合をすべて持つような関係を求める演算.
- 直観的に:
等結合→重複を除去

R		
A	B	C
a	b	c
b	c	d
c	d	e
d	e	f

S		
B	C	D
b	c	f
d	e	a
d	e	c



$R \bowtie S$

R \bowtie S			
A	B	C	D
a	b	c	f
c	d	e	a
c	d	e	c

$R \bowtie S =$

$$\pi_{ABCD}(\sigma_{R.B=S.B \wedge R.C=S.C}(R \times S))$$

Division(商)

- 主要な演算ではないが、有用である場合がある。
Find sailors who have reserved all boats.
“すべてのBをXXするA”や“BをすべてXXするA”
- 定義: Aは二つの属性xとyをもつ; Bは属性yのみを持つ:
 - $A/B = \{\langle x \rangle \mid \exists \langle x, y \rangle \in A \quad \forall \langle y \rangle \in B\}$
 - i.e., A/B は以下のような条件を満たすxタプルから構成される:
Bの各タプルyに対して、Aにタプルxyが存在する。
- 一般に、xとyは属性の集合であってもOK。
- $(A \times B)/B = A$ が成立

例：

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

A

pno
p2

B1

pno
p2
p4

B2

pno
p1
p2
p4

B3

sno
s1
s2
s3
s4

A/B1

sno
s1
s4

A/B2

sno
s1

A/B3

商 A/B の求め方

- STEP 1 : AのうちBに含まれない属性集合 (sno) の一つの組 (aとする) に注目する.
- STEP 2 : Aの属性集合の部分集合でBの属性集合に一致する属性集合Y (pno) を考える.
- STEP 3 : Aにおいて, aに対応するYの組集合Tを考える.
- STEP 4 : TがBを包含しているとき, またそのときのみに, aが最終的な答えに入れれる.

sno	pno	pno
s1	p1	p2
s1	p2	
s1	p3	
s1	p4	
s2	p1	
s2	p2	
s3	p2	
s4	p2	
s4	p4	

A

A/B2

sno
s1
s4

例

Sailors

<u>sid</u>	sname	rating	age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Reserves

<u>sid</u>	<u>bid</u>	day
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Boats

<u>bid</u>	bname	color
101	Interlake	Blue
102	Interlake	Red
103	Clipper	Green
104	Marine	Red

問合せ1:

Boat #103を予約したsailorsの名前は？

- Solution 1: $\pi_{sname}((\sigma_{bid=103} \text{Reserves}) \bowtie \text{Sailors})$
- ❖ Solution 2: $\rho (Temp1, \sigma_{bid=103} \text{Reserves})$
 $\rho (Temp2, Temp1 \bowtie \text{Sailors})$
 $\pi_{sname}(Temp2)$
- ❖ Solution 3: $\pi_{sname}(\sigma_{bid=103}(\text{Reserves} \bowtie \text{Sailors}))$

問合せ2: 赤いboatを予約したsailorsの名前は？

- Boatの色に関する情報はBoatsにしかない

$$\pi_{sname}((\sigma_{color='red'} \text{Boats}) \bowtie \text{Reserves} \bowtie \text{Sailors})$$

❖ 別解 (more efficient?)

$$\pi_{sname}(\pi_{sid}((\pi_{bid} \sigma_{color='red'} \text{Boats}) \bowtie \text{Res} \bowtie \text{Sailors})$$

問合せの最適化 (Query Optimization)

問合せ3:
赤または緑のboatを予約したsailorsの名前は？

- 赤と緑のboatを特定してから、予約したsailorsを探す：

$$\rho (Tempboats, (\sigma_{color='red' \vee color='green'} Boats))$$
$$\pi_{sname}(Tempboats \bowtie Reserves \bowtie Sailors)$$

問合せ4:

赤と緑のboatを予約したsailorsの名前は？

$$\rho (Tempred, \pi_{sid}((\sigma_{color='red'} \text{Boats}) \bowtie Reserves))$$
$$\rho (Tempgreen, \pi_{sid}((\sigma_{color='green'} \text{Boats}) \bowtie Reserves))$$
$$\pi_{sname}((Tempred \cap Tempgreen) \bowtie Sailors)$$

問合せ5:
すべてのboatsを予約したsailorsの名前は？

- 商を利用.

$$\rho (Tempsids, (\pi_{sid,bid} \text{Reserves}) / (\pi_{bid} \text{Boats}))$$

$$\pi_{sname} (Tempsids \bowtie Sailors)$$

- すべての‘Interlake’(boats)を予約したsailorsは？

$$\dots / \pi_{bid} (\sigma_{bname='Interlake'} \text{Boats})$$

まとめ

- 関係モデルの問合せ言語は厳密に定義されている.
- 関係代数は操作・演算中心; 問合せの評価や最適化によく利用される.
- 問合せの表現は複数存在; 最も効率よいものを選びたい(問合せ最適化)

関係モデル・関係データベース概論

馬 強(MA Qiang)

関係データベース

データを表の形で整理する

- 考え方は自然であり、古くからある

1969年にE. F. Coddがこの考え方を計算機上で実装可能なデータモデルとして体系化

- 1981年にチューリング賞を受賞

商用のものやオープンソースソフトウェアなど数多くの関係データベース管理システムが存在

なぜ関係モデルを勉強するのか？

最も広く使われているデータモデルである

- ・ベンダ: IBM, Informix(2001年にIBMが買収), Microsoft, Oracle, Sybase, など
- ・フリーウェア: PostgreSQL, MySQL, Firebird など

より古いモデルの「レガシーシステム」

- ・例 IBM's IMS

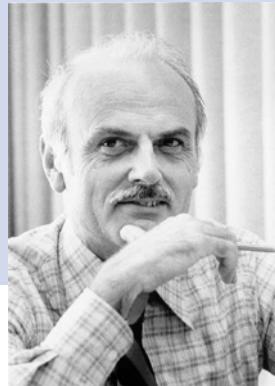
1990年頃 オブジェクト指向データベースが出現

- ・ObjectStore, Versant, Ontos

オブジェクト関係モデル

- ・Informix Universal Server, UniSQL, O2, Oracle, DB2

(1) 関係データモデル



Edgar F. Codd (1923-2003)
1981年チューリング賞受賞

関係

関係データベース (relational database):

- 関係 (*relation*)の集合

関係:

- 行と列から成る表
- 行のことを組(tuple)とも言う. (レコード (record), 行 (row))
- 列のことを属性(attribute), フィールド(field), カラム (column)とも言う.
- フィールドの数 = *degree / arity*, 行の数 = *cardinality*

関係は, 組の集合とみなせる. すなわち, ある関係において一つの組が二回以上現れることはない.

- ただし, 実際の関係データベースシステムでは, 重複を許す場合がある.

関係

属性の定義域：その属性に現れて良い値の集合

関係名

属性

組数 (= 5)

次数 (= 4)

組 (tuple, タプル)

学生番号	学生名	都市	年齢
S1	山田	京都	19
S2	鈴木	大阪	20
S3	小島	奈良	22
S4	武田	京都	18
S5	高木	神戸	21

関係

- 関係 = 組の集合. ゆえに組の順序は任意
- 属性の順序も任意

これら3つの関係
はすべて同じ

学生

学生番号	学生名	都市	年齢
S1	山田	京都	19
S2	鈴木	大阪	20
S3	小島	奈良	22
S4	武田	京都	18
S5	高木	神戸	21

学生-a

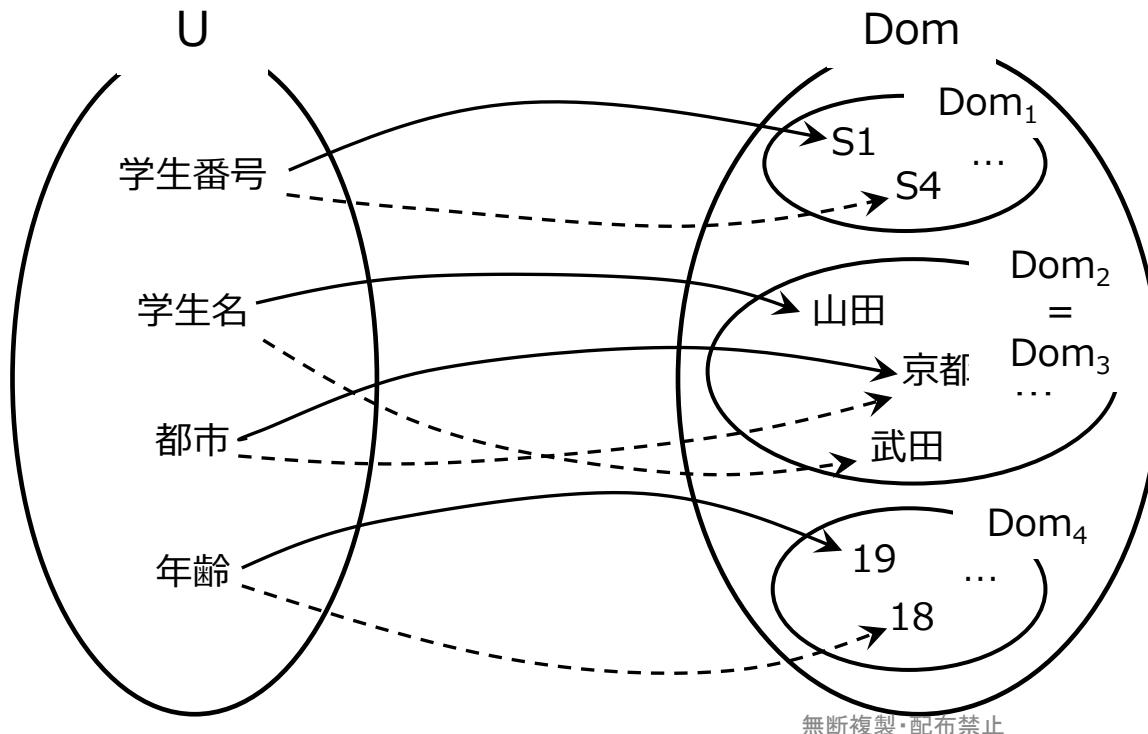
学生番号	学生名	都市	年齢
S3	小島	奈良	22
S2	鈴木	大阪	20
S5	高木	神戸	21
S1	山田	京都	19
S4	武田	京都	18

学生-b

学生名	年齢	都市	学生番号
山田	19	京都	S1
鈴木	20	大阪	S2
小島	22	奈良	S3
武田	18	京都	S4
高木	21	神戸	S5

関係の形式的定義

- 関係は組の集合。組は属性集合 U から値の集合 Dom への写像



属性「学生番号」，「学生名」，「都市」，「年齢」の定義域がそれぞれ $Dom_1, Dom_2, Dom_3, Dom_4$ のとき

実線は組 ("S1", "山田", "京都", 19)を表し，点線は，組("S4", "武田", "京都", 18)を表す。

関係データベース

関係データベーススキーマ:

- 関係スキーマの集合
- 関係間一貫性制約

関係スキーマ

関係スキーマ

- 関係名 R
- 関係の属性集合 $\text{att}(R)$
- 一貫性制約の集合

例

- 関係名： 学生
- 関係の属性集合：
 $\text{att}(\text{学生}) = \{\text{学生番号}, \text{学生名}, \text{都市}, \text{年令}\}$
- 一貫性制約： 属性「年令」の値は、 17 以上の整数のみ
- 一貫性制約の集合を考慮しない場合、 この関係スキーマは、 **学生 (学生番号, 学生名, 都市, 年令)** のように表す。

関係スキーマ

$$(R(A_1, A_2, \dots, A_n), \{\sigma_1, \sigma_2, \dots, \sigma_m\})$$

- 関係スキーマ名
 - 属性集合
 - 一貫性制約の集合
-
- The diagram illustrates the components of a relational schema. At the top, the schema is represented as a tuple: $(R(A_1, A_2, \dots, A_n), \{\sigma_1, \sigma_2, \dots, \sigma_m\})$. Below this, three horizontal lines represent the components: the first line for the schema name, the second for the attribute set, and the third for the constraint set. Blue arrows point upwards from below each line to its corresponding component in the schema tuple.

一貫性制約 (Integrity Constraints)

一貫性制約:

- データベース内のデータが **どのように** 更新されても 真でなければならない条件

一貫性制約の代表的なもの

- 定義域制約 (*domain constraints*)
- 空値に関する制約
- キー制約 (*key constraints*)

一貫性制約は、データベースのデータが 実世界の意味をより忠実に反映する ように、また、データ入力時の 誤りを避ける ために指定する。

関係スキーマのインスタンス

- 関係スキーマ R と $\text{att}(R)$ 上の関係 I があるとき, I が R の一貫性制約をすべて満足するならば, またそのときに限り, I は R の **関係(relation)** または **インスタンス(instance)** であると言う.

関係スキーマとインスタンスの例

スキーマ

インスタンス

(学生 (学生番号, 学生名, 都市, 年齢), $\Sigma(\text{学生})$)

$\Sigma(\text{学生})$ は、次のような一貫性制約を】含む

- σ_1 : 属性「学生番号」に現れる値に重複はない.
- σ_2 : $dom(\text{学生番号}) = \text{String}$
- σ_3 : $dom(\text{学生名}) = \text{String}$
- σ_4 : $dom(\text{都市}) = \text{String}$
- σ_5 : $dom(\text{年齢}) = \text{Integer}$

学生

学生番号	学生名	都市	年齢
S1	山田	京都	19
S2	鈴木	大阪	20
S3	小島	奈良	22
S4	武田	京都	18
S5	高木	神戸	21

関係スキーマとインスタンスの例

スキーマ

成績

(成績 (学生番号, 科目番号, 点数), Σ (成績))

Σ (成績) は、次のような一貫性制約を】含む。

σ_6 : 属性集合 {学生番号, 科目番号} が主キーである.

σ_7 : $dom(\text{学生番号}) = \text{String}$

σ_8 : $dom(\text{科目番号}) = \text{String}$

σ_9 : $dom(\text{点数}) = \text{Integer}$

インスタンス

学生番号	科目番号	点数
S1	J1	75
S1	J2	60
S2	J2	50
S3	J3	90
S3	J4	70
S3	J6	65
S4	J1	50
S4	J2	80
S4	J4	55
S4	J5	75
S4	J6	80

関係スキーマとインスタンスの例

スキーマ

インスタンス

(科目 (科目番号, 科目名, 先生, 単位数), $\Sigma(\text{科目})$)

$\Sigma(\text{科目})$ は、次のような一貫性制約を含む

σ_{10} : 属性集合 {科目番号} が主キーである.

σ_{11} : $dom(\text{科目番号}) = \text{String}$

σ_{12} : $dom(\text{科目名}) = \text{String}$

σ_{13} : $dom(\text{先生}) = \text{String}$

σ_{14} : $dom(\text{単位数}) = \text{Integer}$

科目

科目番号	科目名	先生	単位数
J1	データベース	田中	4
J2	計算理論	佐藤	2
J3	ハードウェア	小林	6
J4	データベース	大野	4
J5	OS	斎藤	5
J6	人工知能	田中	3

定義域制約 (domain constraints)

- 定義域 (domain)
 - 各属性に値として出現することが許されるデータの集合

例： 属性「年令」の定義域は、 17 以上の整数

$$\text{dom}(\text{年令}) = \{ n \mid n \text{ は } n \geq 17 \text{ なる整数} \}$$

空値 (null value; ナル値) 制約

- 値がデータベースに記録されていないことを表わす特別な値

1. 適用不能 (inapplicable)
2. 値は存在するが不明 (unknown)
3. 値が存在しない (not exist)
4. 値が存在するか否かが不明
5. . . .
6. . . .

関係「学生」の例

学生番号	学生名	都市	年令	電話	配偶者	子
S1	山田	京都	19	2 or 3 or 4	花子	3
S2	鈴木	2	20	123-4567	2 or 3 or 4	1 or 2 or 3 or 4
S3	小島	奈良	22	234-5678	月子	一郎
S4	武田	京都	2	345-6789	太郎	2
S5	高木	神戸	21	456-7890	3	1

キー制約 (key constraints)

- 関係スキーマ R において、以下の二つの条件を満足する属性集合 X を、 R の (候補) キー (*(candidate) key*) と呼ぶ：
 - (i) R の任意のインスタンス I において、 I の中のどの二つの異なる組もキーのすべてのフィールドで同じ値を持つことはない。
 - (ii) X の真部分集合で (i) の性質を満足するものは存在しない。
 - 上記の (i)のみが成立し、(ii)は成立しない場合は、スーパー キー (*superkey*) であるという。
 - 関係の候補キーが複数個ある場合は、(DBA (データベース管理者) によって) そのうち一つが主キー (*primary key*) として指定される。

キー制約 (key constraints)

例：{学生番号}は、「学生」のキー

{学生番号, 都市} は、「学生」のスーパーキー

関係スキーマの簡易記法

一貫性制約集合のうち主キーのみを記述する場合の関係スキーマ記述法

学生 (学生番号, 学生名, 都市, 年令)

候補キーの例

- ❖ 三つの大学の「科目」関係スキーマを考える
 - ❖ いずれの大学でも科目番号により科目を一意に決めることができる.
 - ❖ A大学では、科目名と先生を決めると科目は一意に決まる.
候補キーは、{科目番号}と{科目名, 先生}
 - ❖ B大学では、同じ科目名の科目が複数個あることはない.
候補キーは、{科目番号}と{科目名}
 - ❖ C大学では、一人の先生は一科目しか教えない.
候補キーは、{科目番号}と{先生}
- ❖ インスタンス例

A大学

科目番号	科目名	先生	単位数
J1	データベース	田中	4
J2	計算理論	佐藤	2
J3	ハードウェア	小林	6
J4	データベース	大野	4
J5	OS	斎藤	5
J6	人工知能	田中	3

B大学

科目番号	科目名	先生	単位数
J1	データベース	鈴木	4
J2	計算理論	山田	2
J3	ハードウェア	大川	6
J4	OS	斎藤	5
J5	人工知能	鈴木	3

C大学

科目番号	科目名	先生	単位数
J1	データベース	石田	4
J2	計算理論	藤原	2
J3	ハードウェア	林	6
J4	データベース	野口	4
J5	OS	飯田	5
J6	人工知能	中野	3

実体の完全性 (entity integrity)

主キーを構成するどの属性も空値を
値として持つことはできない

一貫性制約について

- データベースインスタンスを検査し一貫性制約を満足しているかどうかを検査できるが、**インスタンスを見ただけでは、ある一貫性制約が存在するかどうかは決して推論することはできない。**
 - 一貫性制約は考え得るすべてのインスタンスに関する述語である
 - インスタンスの例を見て {都市} がキーでないことはわかるが、{学生番号} がキーかどうかはわからない。

一貫性制約 (Integrity Constraints)

- スキーマを定義するときに指定される.
 - 関係が変更されるときに検査される.
 - DBMSは、非合法なインスタンスを許してはならない.
- ❖ 例： 主キー制約の場合
- DBMSは、("S3", "佐藤", "大津", 24) のような組を関係「学生」に挿入することを許さない. (P6)

外部キー制約 (Foreign Keys Constraints)

- ・ 関係間一貫性制約の代表的なもの
- ・ 外部キー (*Foreign key*) : ある関係の属性集合であり、
その値が他の関係のある組を参照するために用いられる
もの
 - 「他の関係」の主キーを用いて参照する
 - 「他の関係」は「ある関係」と同じでも良い

外部キーの例

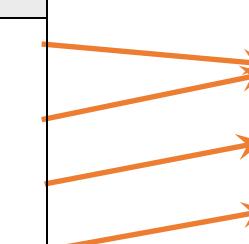
関係： 成績（学生番号, 科目番号, 点数）の属性集合 {学生番号} は、**学生**を参照する外部キー

成績

学生番号	科目番号	点数
S1	J1	75
S1	J2	60
S2	J2	50
S3	J3	90

学生

学生番号	学生名	都市	年令
S1	山田	京都	19
S2	鈴木	大阪	20
S3	小島	奈良	22



参照の完全性 (referential integrity)

ある関係 R_2 が、他の関係 R_1 の主キー (k_1 とする) と対応する外部キー (k_2 とする) を持つなら、 R_2 における k_2 の各値は次のいずれかでなければならない。(R_1 と R_2 は必ずしも異なる必要はない)

- (a) R_1 のある組の k_1 の値と一致する。
- (b) 完全に空値である。 (すなわち k_2 の全属性の値が空値)

すなわち、 dangling tuple (参照先の組がない組) がない。

一貫性制約の執行（外部キーの場合）

成績

学生番号	科目番号	点数
S1	J1	75
S1	J2	60
S2	J2	50
S3	J3	90

学生

学生番号	学生名	都市	年令
S1	山田	京都	19
S2	鈴木	大阪	20
S3	小島	奈良	22

- 存在しない学生番号を値とする「成績」関係の組が挿入さればどうするか？ →拒否する
- 「学生」の組が削除されればどうするか？
 - その組を参照している「成績」のすべての組も削除
 - 「成績」から参照されている「学生」の組は削除を許さない
 - その組を参照している「成績」の組の「学生番号」の値を暗黙値に変更
 - その組を参照している「成績」の組の「学生番号」の値を空値に変更
- 「学生」の組が更新される場合も同様

(2) スキーマ設計

関係DBにおける概念スキーマの設計

方法1: 実体関連モデル→関係モデル

方法2: 従属性を利用した方法(分解法, 合成法)

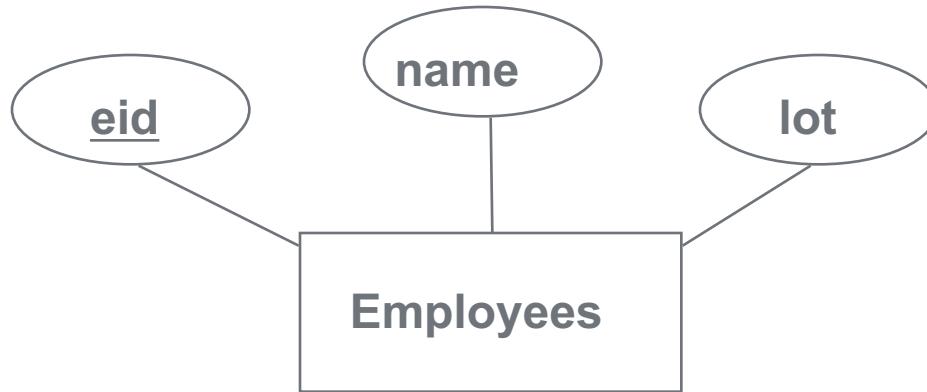
- 必要な属性を全て持つ単一の関係スキーマから成るデータベーススキーマを仮定し、それを改良することによって「正規形」と呼ばれる望ましい性質を持ついくつかの関係スキーマから成る関係データベーススキーマを求める

方法3: 方法1と方法2の併用

実体関連モデル (The Entity-Relationship Model)

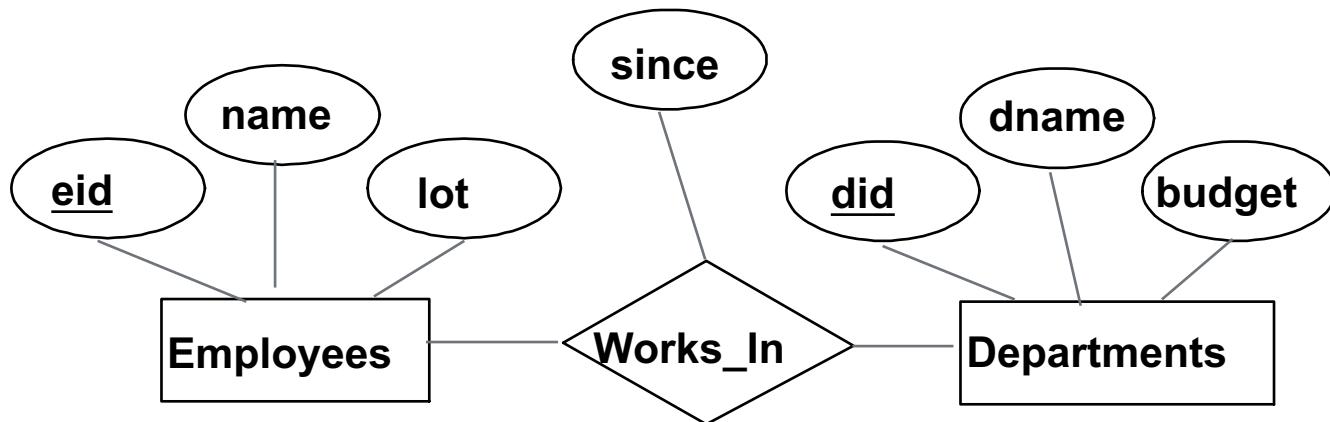
- ERモデルとも言う
- 実世界のデータを
 - 実体 (entity)
 - 関連 (relationship)
 - 属性 (attribute)
- により表現
- データベースの概念設計のためには良く使われる

実体, 実体集合, 属性



- ・ **実体 (entity)** … 実世界において識別可能な物体や事象
- ・ **実体集合 (entity set)** … 同様の実体の集合
(スキーマ的にもインスタンス集合的にも用いられる)
- ・ **属性 (attribute)**
 - 定義域 (domain)
例 : 属性nameの定義域は20文字以下の文字列
- ・ **キー (key)** … 実体を一意に識別する属性の極小集合

関連, 関連集合



- ・ 関連 (relationship) … 複数個の実体間の対応
- ・ 関連集合 (relationship set) … 同様の関連の集まり
(スキーマ的にもインスタンス集合的にも用いられる)
- ・ 記述的属性 (descriptive attribute)
- ・ 関連は参加している実体によって唯一に識別される
 {eid, did} がWorks_Inのキー

関係データベーススキーマの変換

- ある単一の関係スキーマ(R)を変換し別の複数個の関係スキーマの集合($\{R_1, \dots, R_n\}$)に置き換える操作が基本
- 満足すべき基準
 - R と $\{R_1, \dots, R_n\}$ の持つデータ内容は等しい
→ 情報無損失分解
 - R と $\{R_1, \dots, R_n\}$ の保持する意味制約は等しい
→ 従属性の集合が等価 (従属性保存分解)
 - $\{R_1, \dots, R_n\}$ ではデータの冗長性や更新不整合が生じない
→ 正規形

分解法 vs 合成法

分解法

- データベースが対象とする全属性から成る関係を考え、それを射影操作によって分解し、より小さな関係を求めていく手法
- データ内容の保持と正規形に重点

合成法

- 属性間に成立するすべての関数従属性を統合しその両辺に対応する関係スキーマを作っていく手法
- データ内容の保持と意味制約の保持に重点

正規形(Normal Forms)

- スキーマ改良の必要性
 - 冗長
 - 更新不整合
 - 修正不整合
 - 挿入不整合
 - 削除不整合
- 正規形(BCNF,3NF,etc.)の関係であれば、これらの問題を避けることや最小化することが可能

正規形(Normal Forms)

更新不整合などの問題を発生しないように制約条件を与える

- 従属性の概念に基づく
- 冗長を見つけるための関数従属性の役割
- 例： 関係 R(ABC)
 - 関数従属性がなければ、冗長性がない！
 - $A \rightarrow B$ が成立する場合： いくつかの組が同じAの値を持つ；同じBの値も持つ！

条件の程度に応じて、いくつかの正規形が定義

- 第一正規形(first normal form;1NF)の制約にさらに制約を加えるため、高次の正規化と呼ばれる
- 1NF: 各属性の取り得る値は単純値である。

非正規関係

組, 集合, リスト, 関係など構造を持つ値を許す関係

学生番号	学生名	電話番号				
S1	<table border="1"><tr><td>姓</td><td>名</td></tr><tr><td>山田</td><td>太郎</td></tr></table>	姓	名	山田	太郎	{123-456-7890, 234-567-8901}
姓	名					
山田	太郎					
S2	<table border="1"><tr><td>姓</td><td>名</td></tr><tr><td>鈴木</td><td>次郎</td></tr></table>	姓	名	鈴木	次郎	{345-678-9012}
姓	名					
鈴木	次郎					

文字列, 数値, ブール値など構造を持たない値(原子値)のみを許す関係は, 正規化された関係, または第1正規形の関係と呼ぶ。

(3)関係データへの問合せ

関係データベースの問合せ言語

関係データベースの重要な利点: データに
対し単純でしかも強力な問合せが可能

利用者は問合せを直観的に書け, DBMS
はその効率的な処理に責任を持つ

- 重要なこと: 関係問合せの正確な意味論
- 問合せ最適化器が操作の広範な並べ替えをでき,
しかも答が変わらないことを保証する.

DBMSに おける問 合せ

問合せ (query)

- 問合せ言語 (query language)
 - → 関係データベースでは、SQLが標準
- 関係論理 (relational calculus)
- 関係代数 (relational algebra)

DML (data manipulation language)

- 問合せ言語
- データの挿入、削除、修正の機能

応用プログラムからのデータベースの利 用

- 埋込みSQL
- データ副言語 (data sublanguage)と親言語 (host language)
- API ... ODBC, JDBC
- フレームワーク ... Ruby on Rails

SQL

1970年代にIBM (system R) によって開発される

多くのベンダによって使われるため、標準が必要になった

標準:

- SQL-86
- SQL-89 (minor revision)
- SQL-92 (major revision) -> SQL 2
- SQL-99 (major extensions)-> SQL 3
- SQL-2003, SQL-2008, SQL 2011, SQL 2016

SQL 問合せ言語

- 18歳の学生を求める:

```
SELECT *
FROM Students S
WHERE S.age=18
```

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2

name と loginのみを求める場合の一行目 :

```
SELECT S.name, S.login
```

SQLを用いた関係の生成 (定義域制約と空値の制約のみを指定)

```
CREATE TABLE 学生
  (学生番号: CHAR(4),
  学生名: NCHAR(20) NOT NULL,
  都市: NCHAR(10),
  年令: SMALLINT)
```

```
CREATE TABLE 成績
  (学生番号: CHAR(4),
  科目番号: CHAR(5),
  点数: SMALLINT)
```

複数の関係への問合せ

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

S.name	E.cid
Smith	Topology112

組の追加と削除

- ・ 単一の組の挿入:

```
INSERT INTO Students (sid, name, login, age, gpa)
VALUES ('53688', 'Smith', 'smith@ee', 18, 3.2)
```

- ❖ ある条件 (例 name = Smith)を満足するすべての組の削除:

```
DELETE
FROM Students S
WHERE S.name = 'Smith'
```

キー制約 (key constraints)

- 関係スキーマ R において、以下の二つの条件を満足する属性集合 X を、 R の（候補）キー (*candidate key*) と呼ぶ：
 - (i) R の任意のインスタンス I において、 I の中のどの二つの異なる組もキーのすべてのフィールドで同じ値を持つことはない。
 - (ii) X の真部分集合で (i) の性質を満足するものは存在しない。
 - 上記の (i)のみが成立し、(ii)は成立しない場合は、スーパーキー (*superkey*) であるという。
 - 関係の候補キーが複数個ある場合は、(DBA (データベース管理者) によって) そのうち一つが主キー (*primary key*) として指定される。

SQLにおける主キーと候補キー

- 主キー ... PRIMARY KEY
 - 候補キー ... UNIQUE
-
- ❖ ある一人の学生とある一つの科目に対する成績は一つしかない場合
 - ❖ 一人の学生は一つの科目しか登録できず、ある科目を登録したどの二人の学生も同じ点数が付くことはない場合

```
CREATE TABLE 成績  
(学生番号: CHAR(4),  
科目番号: CHAR(5),  
点数: SMALLINT,  
PRIMARY KEY (学生番号, 科目番号))
```

```
CREATE TABLE 成績  
(学生番号: CHAR(4),  
科目番号: CHAR(5),  
点数: SMALLINT,  
PRIMARY KEY (学生番号),  
UNIQUE(科目番号, 点数))
```

外部キー制約 (Foreign Keys Constraints)

- 関係間一貫性制約の代表的なもの
- 外部キー (*Foreign key*) : ある関係の属性集合であり、その値が他の関係のある組を参照するために用いられるもの
 - 「他の関係」の主キーを用いて参照する
 - 「他の関係」は「ある関係」と同じでも良い

SQLにおける外部キー

- 関係「学生」に現れる学生のみが科目の登録を許される。

CREATE TABLE 成績

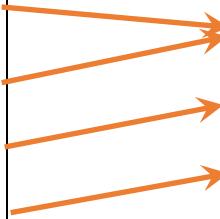
(学生番号 CHAR(4), 科目番号 CHAR(5), 点数 SMALLINT,
PRIMARY KEY (学生番号, 科目番号),
FOREIGN KEY (学生番号) REFERENCES 学生)

成績

学生番号	科目番号	点数
S1	J1	75
S1	J2	60
S2	J2	50
S3	J3	90

学生

学生番号	学生名	都市	年令
S1	山田	京都	19
S2	鈴木	大阪	20
S3	小島	奈良	22



参照の完全性 (referential integrity)

ある関係 R_2 が、他の関係 R_1 の主キー (k_1 とする) と対応する外部キー (k_2 とする) を持つなら、 R_2 における k_2 の各値は次のいずれかでなければならない。 (R_1 と R_2 は必ずしも異なる必要はない)

- (a) R_1 のある組の k_1 の値と一致する。
- (b) 完全に空値である。 (すなわち k_2 の全属性の値が空値)

すなわち、dangling tuple (参照先の組がない組) がない。

SQLにおける参照制約

- SQL 92 と SQL 99 は、削除と更新に関し、四つのオプションすべてをサポートしている。
 - デフォルトは、NO ACTION (削除/更新はできない)
 - CASCADE (削除された組を参照する組もすべて削除する)
 - SET NULL / SET DEFAULT (参照している組の外部キーの値を設定する)

(4) ビュー

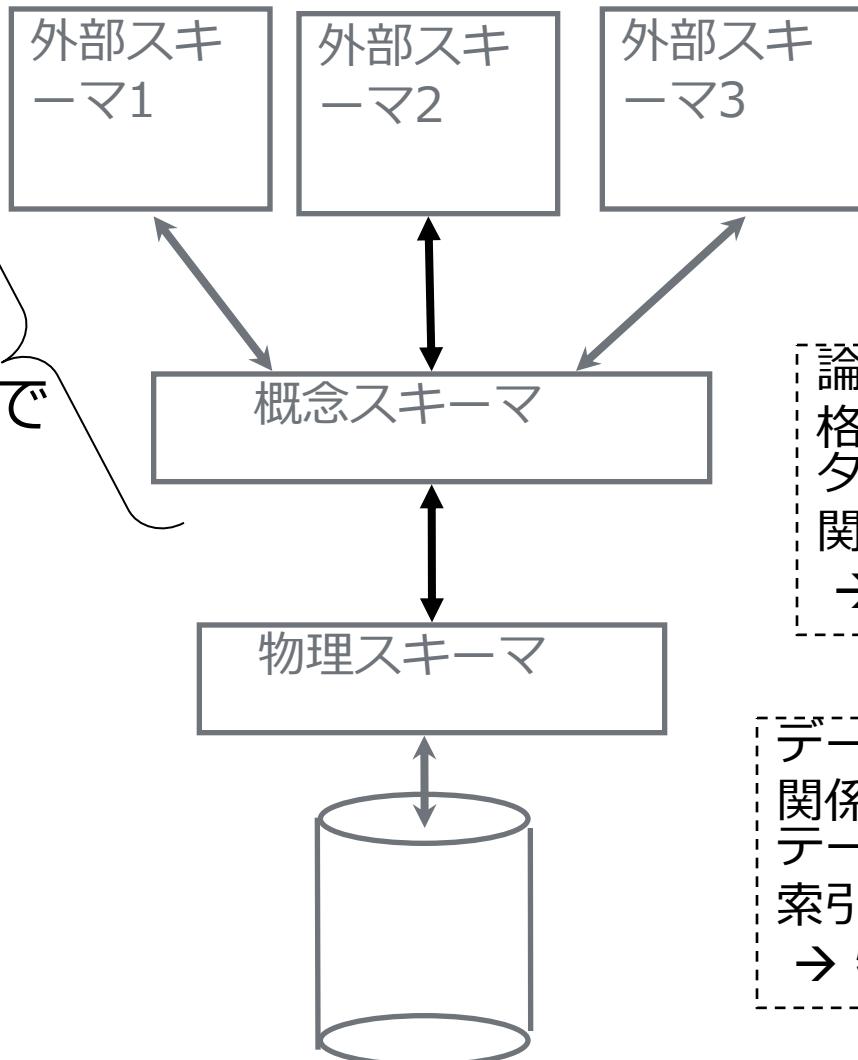
ビュー (Views)

- ビューは単なる関係。しかし、組の集合を格納するのではなく、定義を格納する。
- ビュー定義文

```
CREATE VIEW YoungActiveStudents (name, grade)
    AS SELECT S.name, E.grade
    FROM Students S, Enrolled E
    WHERE S.sid = E.studid and S.age<21
```

復習： DBMSにおける抽象度のレベル

DDLで
定義

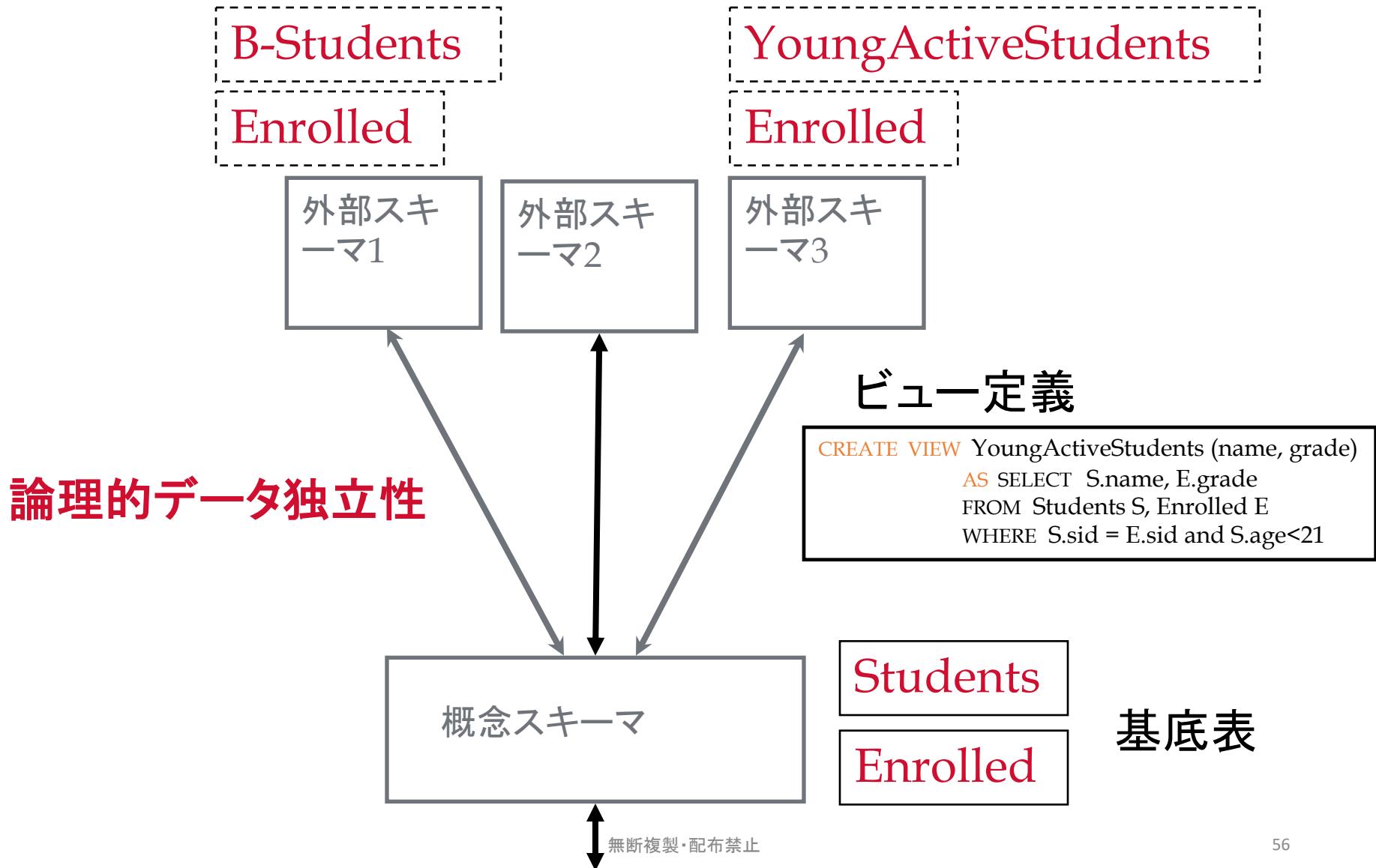


各利用者（のグループ）向けに再構成されたデータをDBMSのデータモデルで記述
ビュー（view）と概念スキーマの関係から成る。ビューは関係のようなものだが、データは格納されていない。

論理スキーマとも呼ばれる。
格納されているデータをDBMSのデータモデルにより記述
関係およびフィールドの選択
→ 概念データベース設計

データ格納に関する詳細
関係が実際にどのようにディスクやテープに格納されるかを記述
索引（index）の指定など
→ 物理データベース設計

外部スキーマとビュー



ビュー

学生「小島」のビュー



小島履修状況

科目番号	科目名	先生	単位数	点数
J3	ハードウェア	小林	6	90
J4	データベース	大野	4	70
J6	人工知能	田中	3	65

外部スキーマ

概念スキーマ

学生

学生番号	学生名	都市	年齢
S1	山田	京都	19
S2	鈴木	大阪	20
S3	小島	奈良	22
S4	武田	京都	18
S5	高木	神戸	21

成績

学生番号	科目番号	点数
S1	J1	75
S1	J2	60
S2	J2	50
S3	J3	90
S3	J4	70
S3	J6	65
S4	J1	50
S4	J2	80
S4	J4	55
S4	J5	75
S4	J6	80

無断複製・配布禁止

科目

科目番号	科目名	先生	単位数
J1	データベース	田中	4
J2	計算理論	佐藤	2
J3	ハードウェア	小林	6
J4	データベース	大野	4
J5	OS	斎藤	5
J6	人工知能	田中	3

ビュー

先生「田中」のビュー



J1 成績

学生番号	学生名	点数
S1	山田	75
S4	武田	50

外部スキーマ

J6 成績

学生番号	学生名	点数
S3	小島	65
S4	武田	80

概念スキーマ

学生

学生番号	学生名	都市	年齢
S1	山田	京都	19
S2	鈴木	大阪	20
S3	小島	奈良	22
S4	武田	京都	18
S5	高木	神戸	21

成績

学生番号	科目番号	点数
S1	J1	75
S1	J2	60
S2	J2	50
S3	J3	90
S3	J4	70
S3	J6	65
S4	J1	50
S4	J2	80
S4	J4	55
S4	J5	75
S4	J6	80

無断複製・配布禁止

科目

科目番号	科目名	先生	単位数
J1	データベース	田中	4
J2	計算理論	佐藤	2
J3	ハードウェア	小林	6
J4	データベース	大野	4
J5	OS	斎藤	5
J6	人工知能	田中	3

ビューと安全性

- ビューは、必要な情報（またはその要約）を見せ、もとの関係の詳細は隠蔽するために使える。
- 例：
学生のname と age は存在するが、gpaは隠蔽されているような
ビューを作成し、学生にはStudents表ではなく、このビューをアクセ
スすることを許す。

ビュー

- ❖ 問合せやビュー定義文の中で基底表(base table)と同様に使える.
- ❖ ビューは, **DROP VIEW** コマンドを用いて破壊できる.
 - あるビューの元となる基底表が **DROP TABLE** で破壊されたらどうなるか?
 - **DROP TABLE 表名 RESTRICT**
ビューや一貫性制約が表を参照していないければ表を破壊
 - **DROP TABLE 表名 CASCADE**
表を参照しているビューや一貫性制約も破壊

関係の破壊と変更

DROP TABLE Students

- Students関係を破壊する。スキーマ情報と組がともに削除される。

ALTER TABLE Students

 ADD COLUMN firstYear: integer

- ❖ 新しいフィールドを追加してStudents のスキーマを変更する。現在の各インスタンスの新しいフィールドの値は空値(null) になる。

関係データモデル：データをテーブルで表現

シンプル、直感的モデルであり、最もよく利用されている

一貫性制約はDBAが実際のアプリケーションに応じて定義する。
DBMSはその違反をチェックする。

- 二つ重要な一貫性制約：主キーと外部キー
- ドメイン制約は常にある

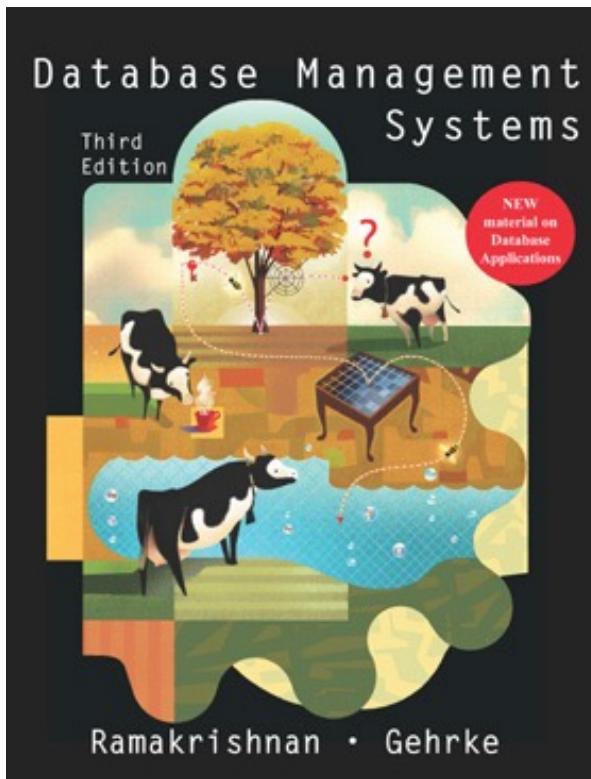
まとめ

データベース

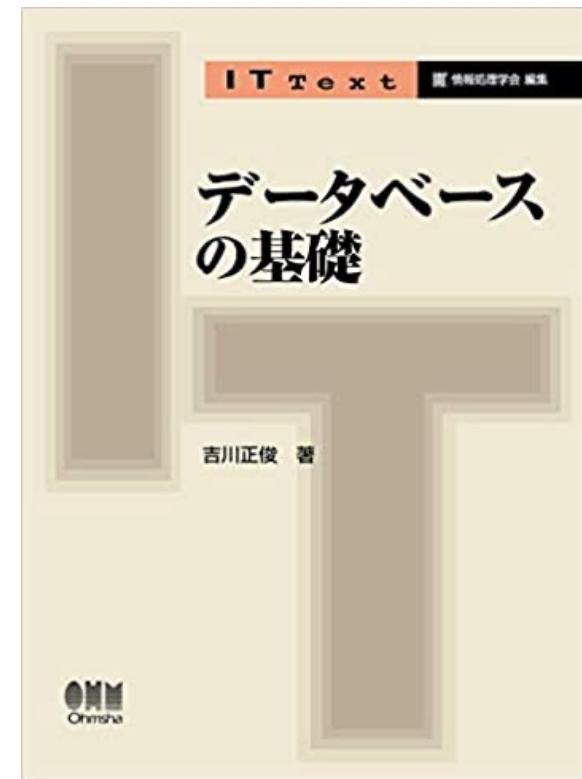
担当教員：馬

qiang@i.Kyoto-u.ac.jp

参考図書



[Raghu Ramakrishnan](#) and [Johannes Geheke](#):
"Database Management Systems, Third Edition",
McGraw-Hill, 2002.
ISBN: 0071230572 (paperback)



吉川正俊 『データベースの基礎』
(オーム社) ISBN:978-4-274-22373-0

成績評価：
小テスト, レポート, 期末試験をもとに
総合的に判断する.

スケジュール

イントロダクション(1回)

関係データベース(2回)

関係データベースの形式的操作体系とSQL(4回)

スキーマ設計法(3回) :

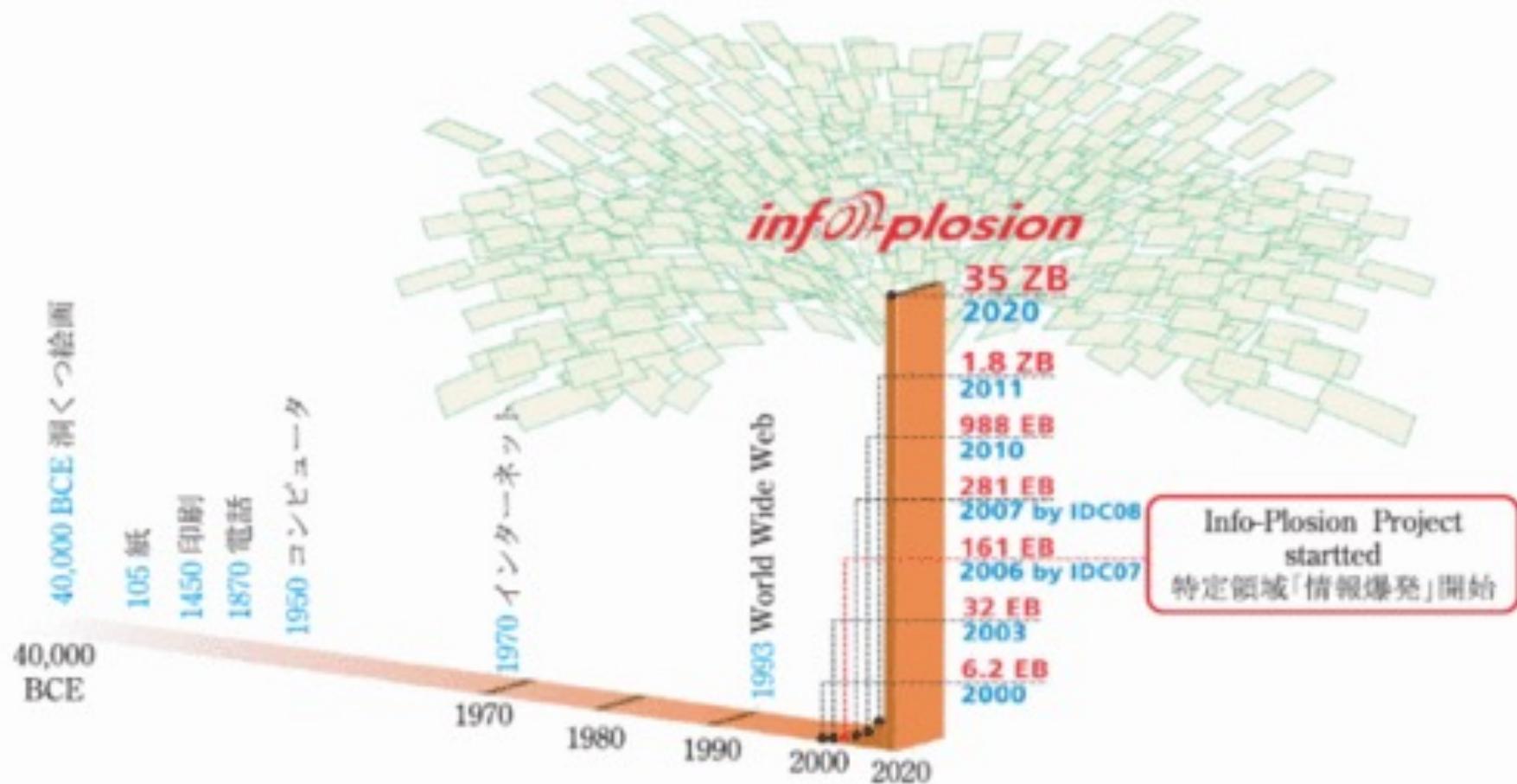
記憶装置およびファイル編成法(2回) :

トランザクション(2回) :

フィードバック(1回)

1. 概論

データの爆発的増加

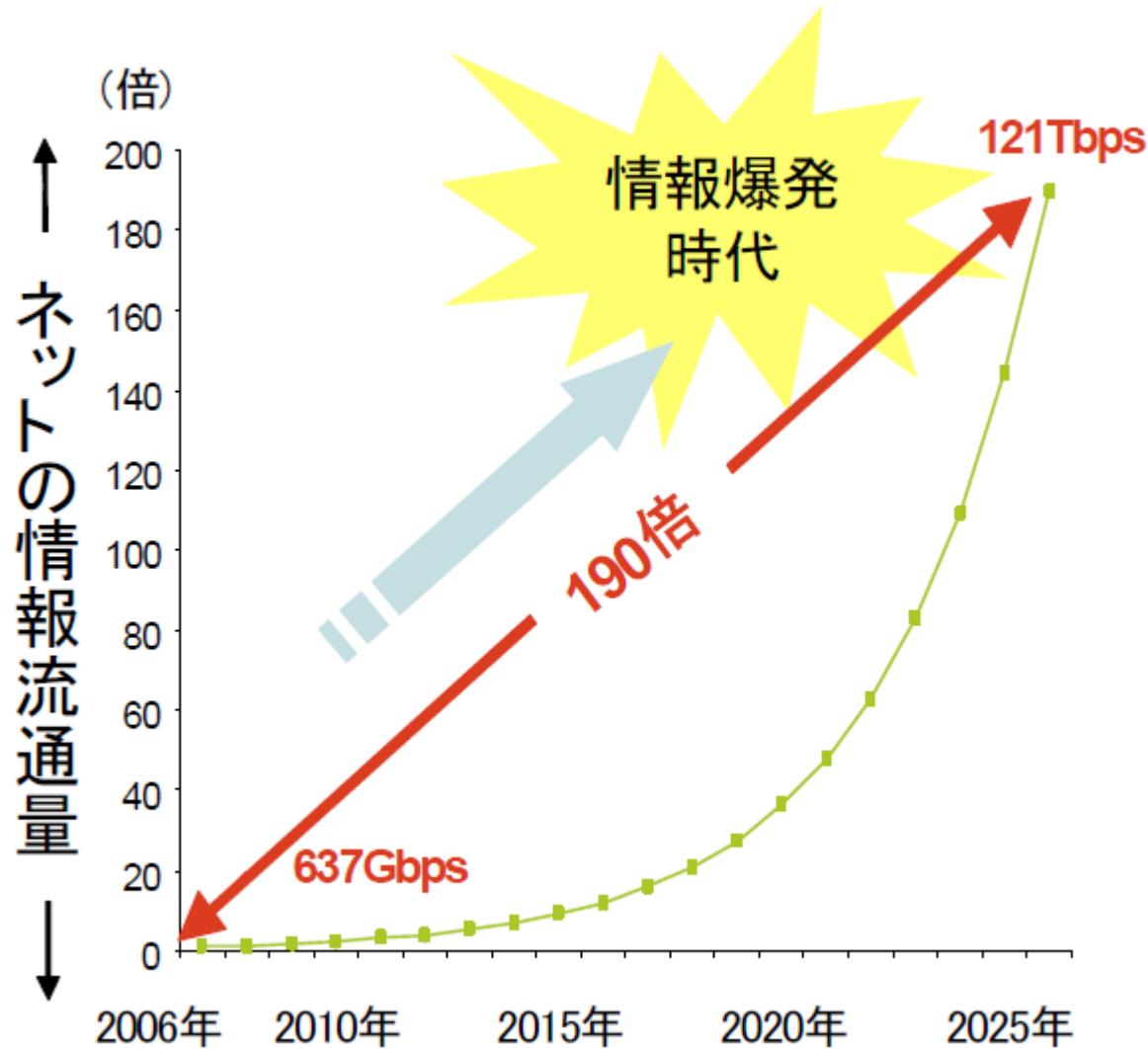


情報爆発 (出典: Horison Information Strategies, cited from Storage New Game New Rules, p. 34 (www.horison.com), IDC, The Diverse and Exploding Digital Universe 2020 (<http://www.emc.com/collateral/demos/microsites/idc-digital-universe/iview.htm>))

出典: 喜連川優「情報爆発のこれまでとこれから」, 電子情報通信学会誌, Vol.94, No8, 2011
無断配布・複製は禁止

データの爆発的増加

平成20年6月4日、経済産業省商務情報政策局、「経済産業省の情報政策について」
http://home.jeita.or.jp/is/committee/infoterm/pdf/080604festival_meti.pdf



データ管理のために必要な機能

多様で大規模なデータを対象とする

データの正しさを管理する

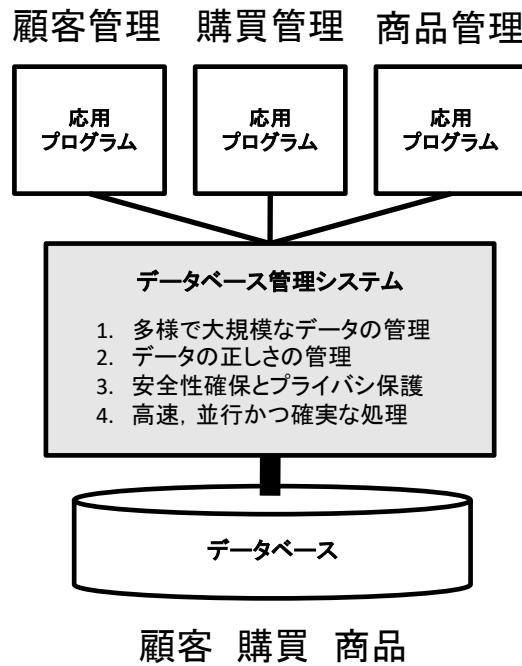
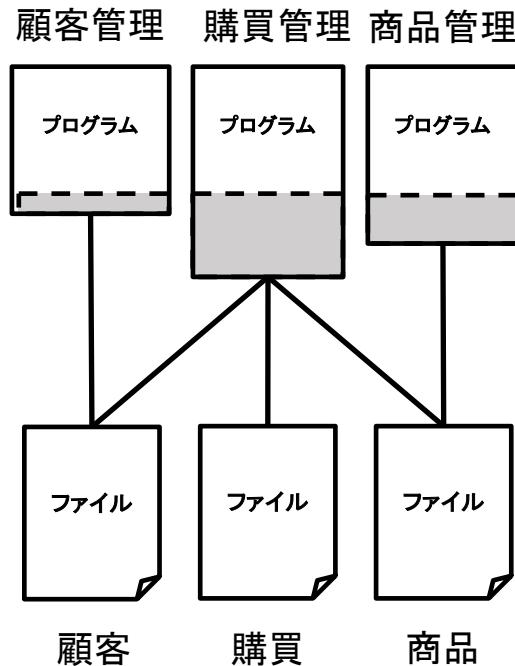
データの安全性確保とプライバシ保護を行う

高速、並行、かつ確実に処理する

表計算ソフトとデータベースシステムの違い

	表計算ソフト	データベース システム
1 多様で大規模なデータ	×	○
2 データの正しさを管理	△	○
3 データの安全性、プライバシ保護	×	○
4 高速かつ並行に処理	×	○

ファイルとデータベースの違い



DBMSとデータベース

データベース管理システム

- (Database Management Systems; DBMS)
- 大量のデータを格納し、一人または複数の利用者による検索(retrieval), 更新(update)およびそれに付随する機能を提供するシステム

データベース

- ある意図をもって集められた計算機可読形式のデータ
- 狹義には、DBMSで管理されているデータ

学問としてのデータベース

データベースは計算機科学の縮図

- プログラミング言語, O.S., 並行プログラミング, データ構造, アルゴリズム, 理論, 並列, 分散システム, 利用者インターフェース, 人工知能, . . .

社会との密接な関連

- 情報公開, 安全性, トレーサビリティ, データ爆発, . . .

実学としてもきわめて重要

- ほぼすべての組織がデータベースを利用
- データベースベンダー:
 - Oracle, IBM, Microsoft, 日立, ...
- オープンソース
 - MySQL, PostgreSQL, Firebird, ...

多様なデータベース

関係データベース

- 企業の財務システム、人事管理システム、顧客関係管理システム
 - [Oracle](#), [SQL Server](#), [DB2](#), [MySQL](#), [PostgreSQL](#)
- メインメモリデータベース (Oracle TimesTen)

スケーラビリティ優先のデータベース

- Sharding ... 多数(数十から数百)のRDBMSによるデータ共有
 - データベース間のインスタンスの結合、集約、大域的な唯一の二次索引、大域的なストアドプロシージャなどの関係データベース機能を期待できないという大きい制約の下でのプログラミングモデル
- Shardingという用語に明確な定義はないことに注意
- MySQLのshardingは、複数のDBMSの個々が分割データベースを管理
- スケーラブルなキーバリュー型データストア：例：[Amazon SimpleDB](#)
- 結果的に整合性のある読み込み-読み込みパフォーマンスを最大化。ただし、最近完了した書き込みの結果を反映しない可能性がある。

単純な構造化記憶

- 構造化された永続記憶、単純な問合せや索引付けは必要だが、RDBMSほどの機能、コスト、複雑性は必要とせず、スケーラビリティも不要。
- 例：[Berkeley DB](#)(基本ファイル構造は、B-treeとextensible hash)

特定の目的に最適化されたデータベース

- 例：[StreamBase](#): 株価データ管理、実時間損益管理など複雑なイベント処理 (complex event processing)に特化

基幹系処理と情報系処理

基幹系処理

- 組織の基幹業務を遂行するためのデータ処理
- 銀行 ... 顧客による口座への出入、送金、貸出など
- Amazon ... 顧客による購買に伴う伝票処理、クレジットカード引き落とし、ポイント管理など

情報系処理

- 基幹系処理により日々大量に発生するデータを蓄積し、それを解析することにより組織の意志決定を行うための処理
- 銀行 ... 顧客の過去の残高履歴から定期預金を勧誘することなど
- Amazon ... 販売促進のために顧客の購買履歴を解析し次に購入する可能性がある商品を推薦

基幹系処理と情報系処理

	基幹系処理	情報系処理
1処理あたりに必要なデータ量	少ない	膨大
処理の複雑さ	単純	複雑
検索/更新	更新主体	検索主体
処理件数	多い	少ない
結果の厳密性	必要	近似値でよい場合もある
オンラインデータベース処理	OLTP	OLAP

OLTP(Online Transaction Processing)

OLAP(Online Analytical Processing)

データ管理

関係データベース管理システム (Relational Database Management Systems; RDBMSs)

この講義の中心的な話題

- データモデル
- データベース設計
- ファイル編成法
- データ問合せ言語
- トランザクション管理

DBMSでのデータの記述と格納

データモデル

- 多種多様で相互に複雑に関連しているデータを統一的に整理し計算機での格納や操作を容易にするために、データの構造、操作、制約などを抽象化して表現するモデル
- 格納に関する低レベルの多くの事項を隠した、高レベルのデータ記述のための構成要素の集まり

これまでに非常に多くのデータモデルが提案された

- 関係データモデル(relational data model)
- 意味的データモデル(semantic data model)
 - より抽象的で高レベルなデータモデル
 - 例：実体関連モデル（ERモデル）
- キーバリューモデル (key-value model)
- オブジェクト指向モデル(object-oriented model)
- オブジェクト関係モデル(object-relational model)
- 網モデル (network model)

関係の例

大学の「学生」

学生番号	学生名	都市	年令
S1	山田	京都	19
S2	鈴木	大阪	20
S3	小島	奈良	22
S4	武田	京都	18
S5	高木	神戸	21

インターネットショップの「顧客」

顧客番号	顧客名	都市	年令
S1	山田	京都	19
S2	鈴木	大阪	40
S3	小島	奈良	32
S4	武田	京都	18
S5	高木	神戸	51

これらのテーブルの問題点は？

関係モデル

関係 (relation) ... 表のこと. レコードの集合

レコード (record) ... 表の各行のこと.

スキーマ (schema) ... 表の枠組み.

- 関係モデルでは、関係のスキーマは、
 - 関係の名前
 - 各フィールド(field), (または属性(attribute), 列(column)) の名前
 - 各フィールドの型
 - から成る。本当はさらに
 - 一貫性制約 (integrity constraints)もある。
 - 例: 「各学生の学生番号の値は唯一でなければならない」

インスタンス (instance) ... 表の内容, データそのもの

実体関連モデル (ERモデル)

この世に存在するすべてのデータを二種類に分類、整理

- 実体 (entity)
 - 学生、教員、科目、教室など
- 関連 (relationship)
 - 実体間の関連
 - 学生が科目に登録すること、教員が科目を教えること、科目のために教室を使用すること

スキーマとインスタンス

- ・関係モデルの場合

顧客 (顧客 ID, 顧客名, 年齢, email)

購入履歴 (顧客 ID, 商品 ID, 数量, 購入月日)

商品 (商品 ID, 商品名, 単価)

スキーマ

顧客

顧客 ID	顧客名	年齢	email
c1	Yamada	25	yamada@abc
c2	Suzuki	38	suzuki@xyz

購入履歴

顧客 ID	商品 ID	数量	購入月日
c1	p1	3	06-29
c2	p2	2	07-04
c2	p3	1	07-05

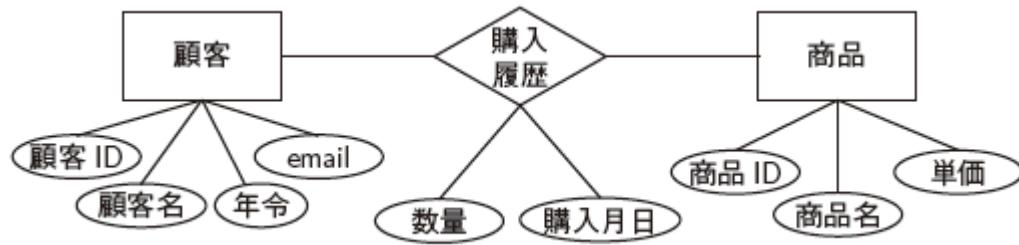
商品

商品 ID	商品名	単価
p1	Super Wet	300
p2	Ginmugi	150
p3	Nourei	200

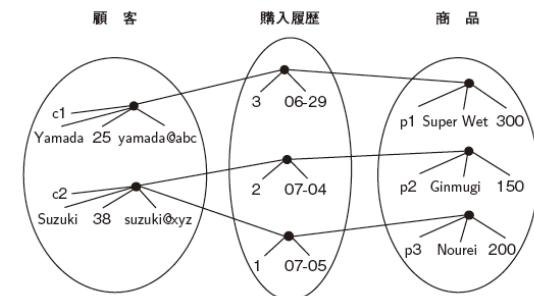
インスタンス

スキーマとインスタンス

- ・実体関連モデルの場合



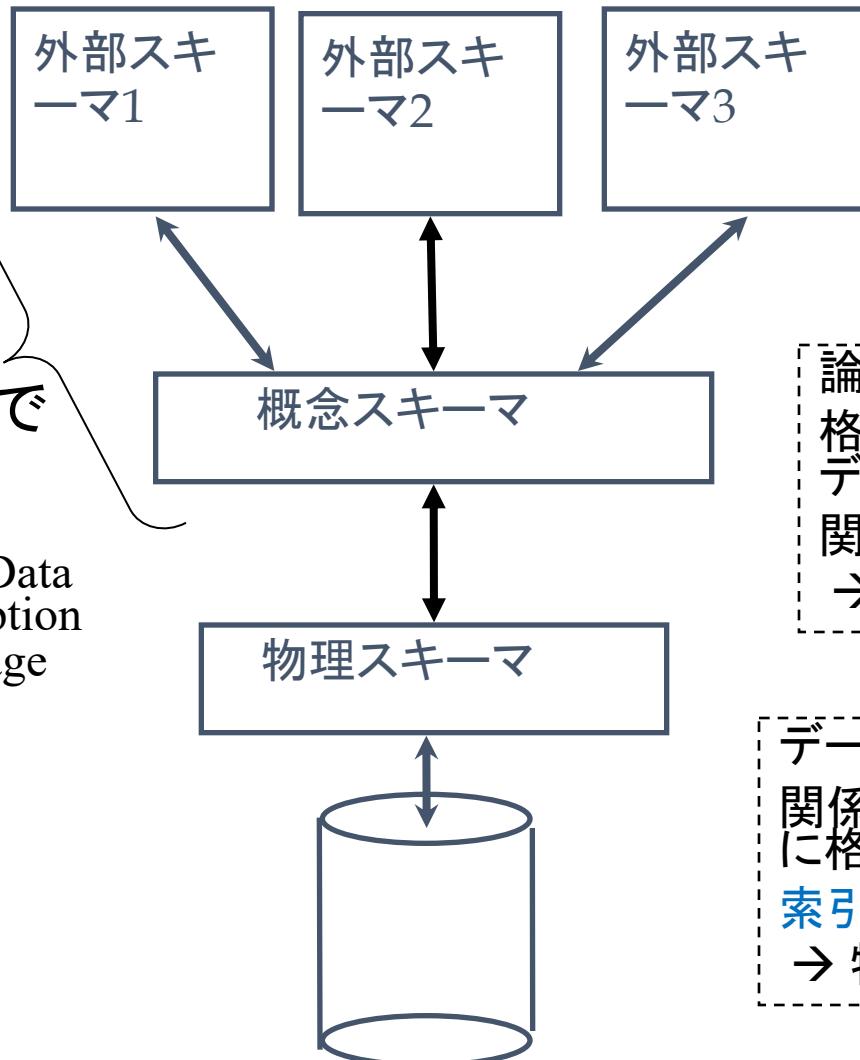
スキーマ



インスタンス

DBMSにおける抽象度のレベル

DDLで定義
DDL: Data Description Language

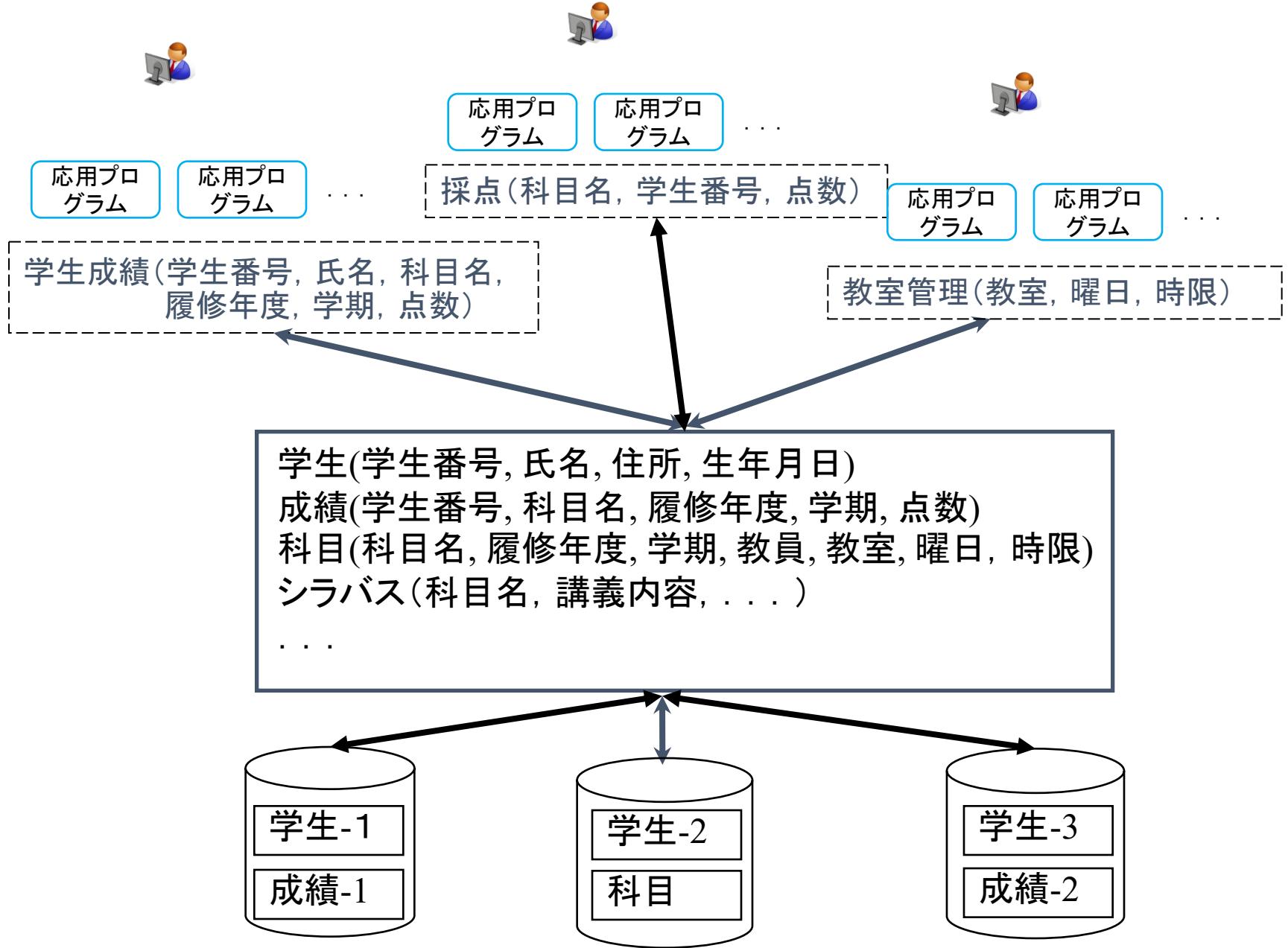


ANSI/SPARCモデル

各利用者(のグループ)向けに再構成されたデータをDBMSのデータモデルで記述
ビュー (view)と概念スキーマの関係から成る。ビューは関係のようなものだが、データは格納されていない。

論理スキーマとも呼ばれる。
格納されているデータをDBMSのデータモデルにより記述
関係およびフィールドの選択
→ 概念データベース設計

データ格納に関する詳細
関係が実際にどのようにディスクやテープに格納されるかを記述
索引 (index)の指定など
→ 物理データベース設計



データ独立性

応用プログラムが、データの構造化および格納の方法から防護されていること。

- 三レベルのデータ抽象化、特に、概念スキーマと外部スキーマにより実現されている。

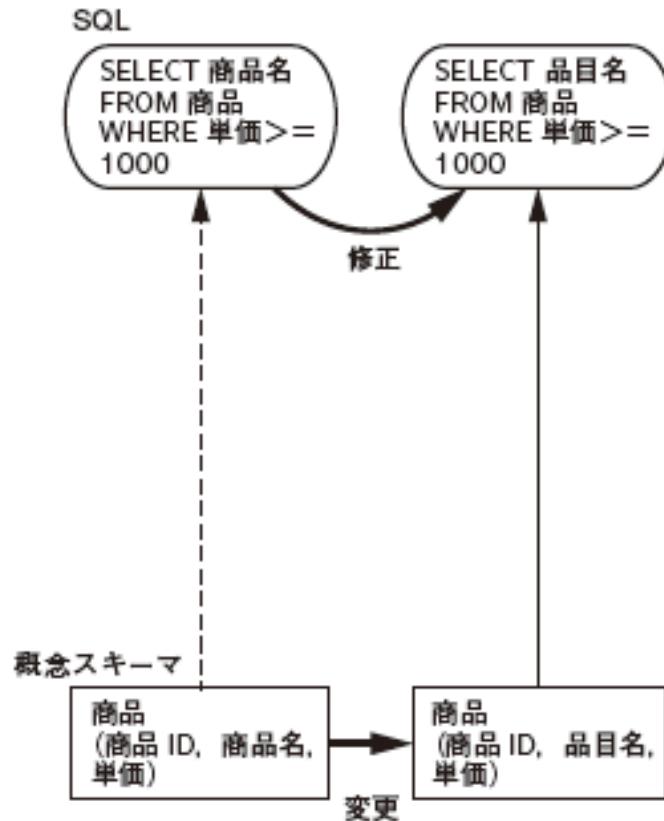
ビューの元になる概念スキーマのデータが再編成されれば、以前と同じ関係が計算されるようにビュ一定義を修正することができる。

- 論理的データ独立 (logical data independence)

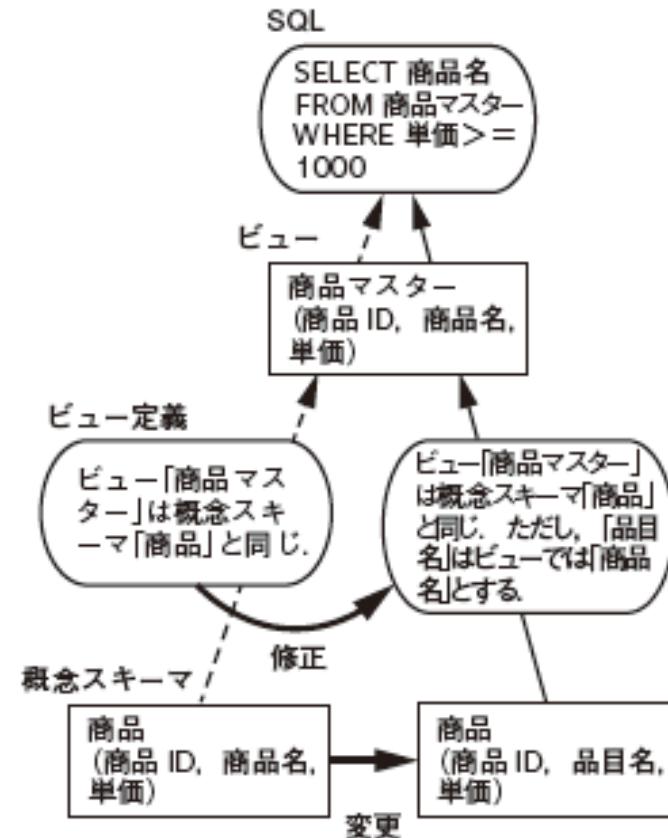
物理的な格納に関する詳細から概念スキーマが保護されていること。

- 物理的データ独立 (physical data independence)

論理的データ独立性



論理的データ独立性なし



論理的データ独立性あり

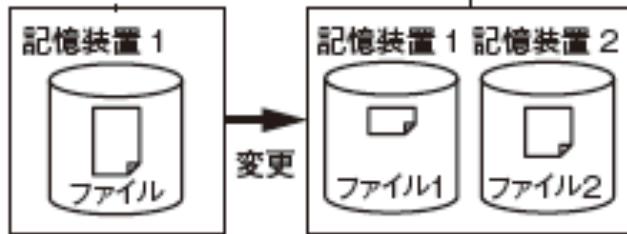
物理的データ独立性

概念スキーマ



変更

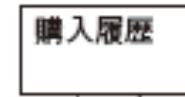
内部スキーマ



変更

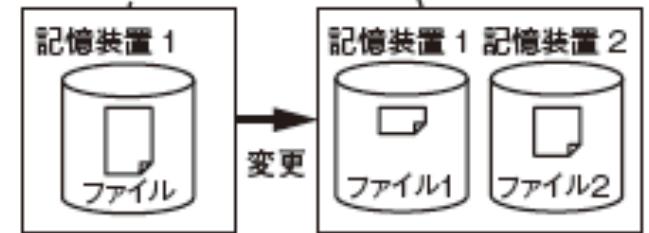
物理的データ独立性なし

概念スキーマ



DBMS

内部スキーマ



変更

物理的データ独立性あり

DBMSにおける問合せ

問合せ (query)

- 問合せ言語 (query language)
 - → 関係データベースでは、SQLが標準
- 関係論理 (relational calculus)
- 関係代数 (relational algebra)

DML (data manipulation language)

- 問合せ言語
- データの挿入、削除、修正の機能

応用プログラムからのデータベースの利用

埋込みSQL

- データ副言語 (data sublanguage) と 親言語 (host language)

API ... ODBC, JDBC

フレームワーク ... Ruby on Rails

平均点最高の学生の氏名
を知りたい



応用プログラム

応用プログラム

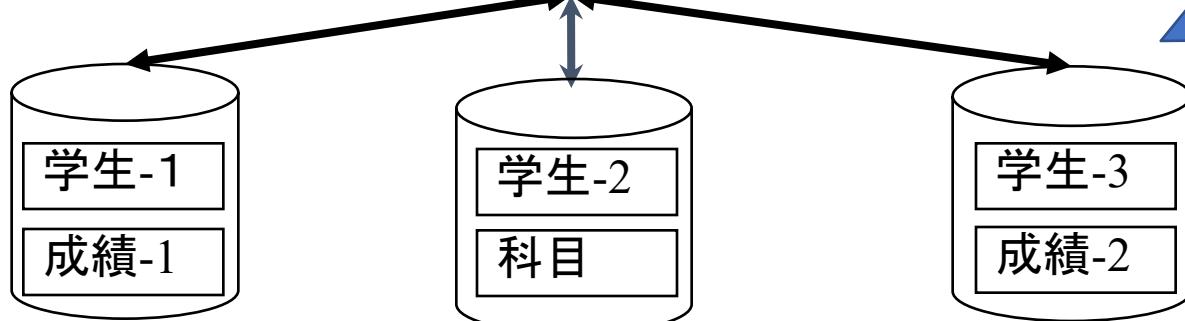
...

採点(科目名, 学生番号, 点数)

高水準言語

- 非手続き的
- whatを指定

学生(学生番号, 氏名, 住所, 生年月日)
成績(学生番号, 科目名, 履修年度, 学期, 点数)
科目(科目名, 履修年度, 学期, 教員, 教室, 曜日, 時限)
シラバス(科目名, 講義内容, . . .)
...



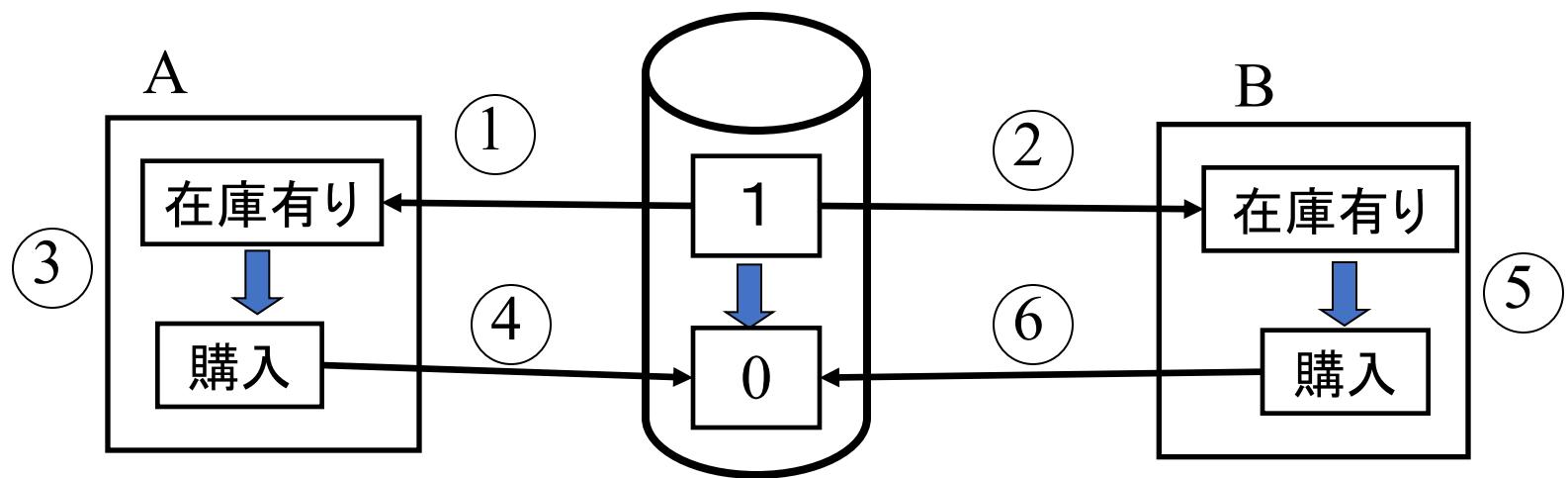
低水準言語

- 手続き的
- howを指定

1. 成績-1, 2から学生ごとの平均点を計算する
2. その結果をソートし, 最高点を取った学生の学生番号を求める
3. 学生-1, 2, 3を順に探しその学生番号の学生の氏名を見つける

トランザクション管理

例：多くの顧客が同じ商品を購入する場合



うまく管理しなければ架空の商品を売ることになる。

トランザクション管理

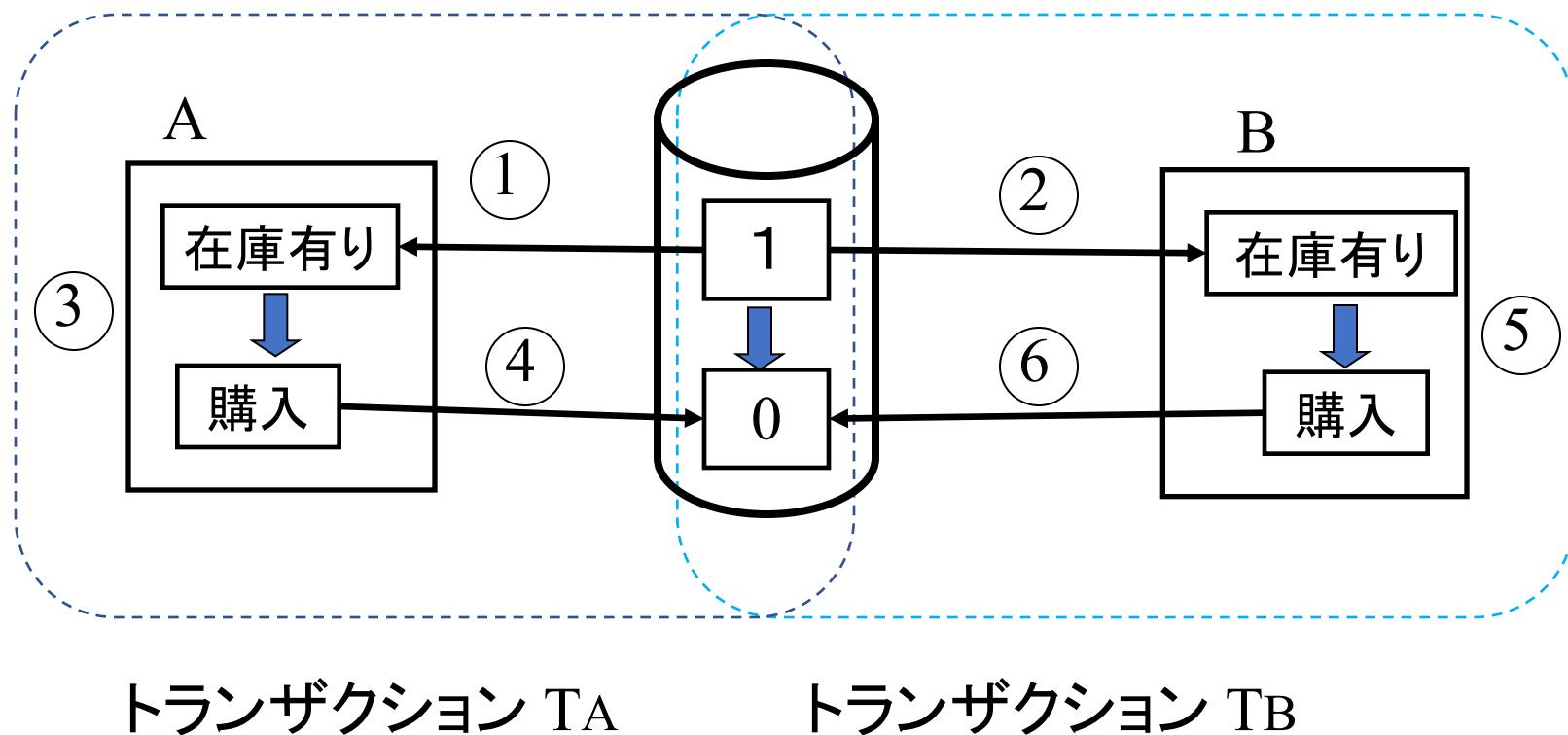
トランザクション (transaction)

- DBMS内での一つの利用者プログラムのある**1回の実行**
- DBMSから見える**変更の基本単位**. すなわち, 部分的なトランザクションの実行は許されない.
- トランザクションのグループを実行した結果は, そのグループ内のトランザクションがある直列順序で実行した結果と等価でなければならない.

ACID (Atomicity:原子性, Consistency:一貫性, Isolation:独立性, Durability:永続性)

トランザクション管理

例：多くの顧客が同じ商品を購入する場合



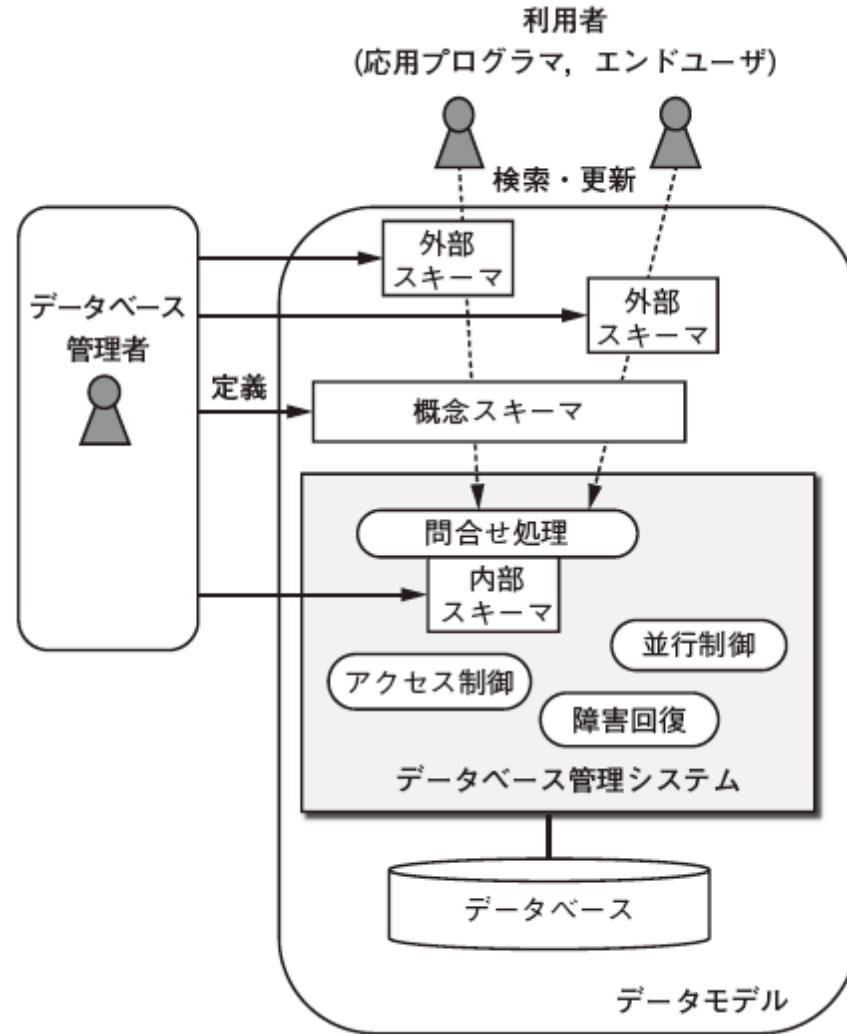
不完全なトランザクションとシステム障害

DBMSは、システム障害などにより不完全なまま終了したトランザクションが行ったデータ修正は、データベースから取り除く必要がある。

例：「A から Bへの 100万円の送金」というトランザクションは、以下の二つの操作からなる。

- 1. $A \leftarrow A - 100\text{万}$
- 2. $B \leftarrow B + 100\text{万}$
- 操作1終了後、システム障害が生じトランザクションが中止された場合は、操作1も取り消されなければならない。

データベース管理システム



データベースに関する人々

データベース実装者 (database implementors)

末端利用者 (end users)

- 多くの末端利用者は、データベース応用プログラマが書いた応用を使うだけ。

データベース応用プログラマ (database application programmers)

- 親言語、データ言語、ソフトウェアツールなどを用いて末端利用者がデータへアクセスするためのパッケージを作成

データベース管理者 (database administrator (DBA))

熟練者の人件費 → 管理の自動化

DBAが責任を持つ作業

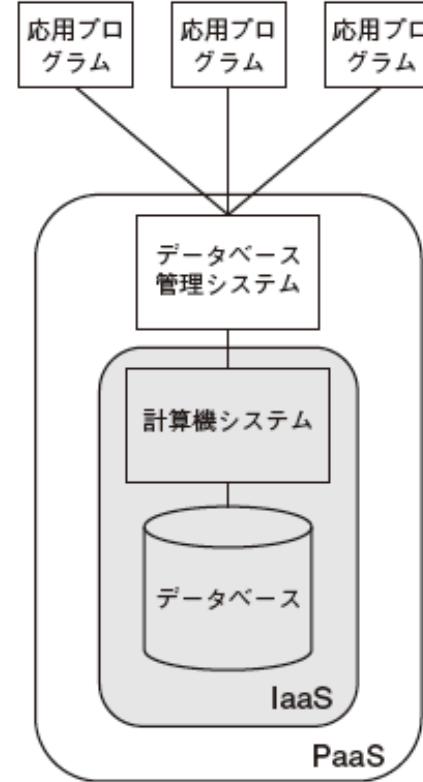
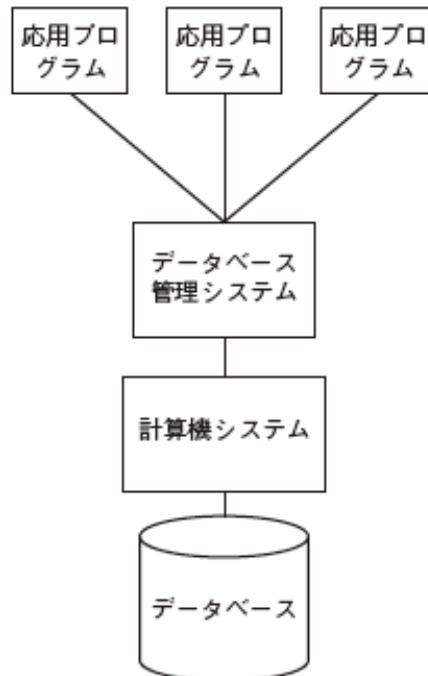
概念および物理スキーマの設計

安全性と認証の管理

データ可用性と障害からの回復

データベースチューニング

データベース管理システムの提供形態



オンプレミスデータベース

クラウドデータベース

課題

身边にデータベースを用いたシステムをひとつ選んで、そのシステムの概要を説明しなさい。また、そのシステムにおけるデータベースの役割について説明しなさい。

400文字以内

次回(4/17 (月) 12:00)までPandAにて提出。

関係モデル

次回

SQL

関係データモデルとSQLの用語

関係データモデル	SQL	備考
関係(relation)	表(table)	
属性(attribute)	列(column)	表の列は順序付けられている。
組(tuple)	行(row)	表の中で重複した行の存在を許す。 また、問合せ結果表の行は順序づけられていてもよい。
定義域(domain)	データ型(data type), 定義域(domain)	
	ナル値(null value)	

関係データモデルとSQLの用語

- 関係(relation), 表(table)
- 属性(attribute), 列(column)
- 組(tuple), 行(row)

関係, 表

関係名 → 学生

属性, 列 属性名

学生番号	学生名	都市	年令
S1	山田	京都	19
S2	鈴木	大阪	20
S3	小島	奈良	22
S4	武田	京都	18
S5	高木	神戸	21

組, 行

SQLのデータ定義言語

「学生」表の作成

```
CREATE TABLE 学生 (
    学生番号 VARCHAR(4),
    学生名 NVARCHAR(20) NOT NULL,
    都市 NVARCHAR(10),
    年令 INTEGER,
    PRIMARY KEY (学生番号))
```

主キーの指定,
主キーは暗黙的にNOT NULLが
付いているとみなされる。

```
CREATE TABLE 科目 (
    科目番号 VARCHAR(5),
    科目名 NVARCHAR(15) NOT NULL,
    先生 NVARCHAR(10),
    単位数 INTEGER,
    PRIMARY KEY (科目番号))
```

ナル値を許さない

```
CREATE TABLE 成績 (
    学生番号 VARCHAR(4),
    科目番号 VARCHAR(5),
    点数 INTEGER,
    PRIMARY KEY (学生番号, 科目番号)
    FOREIGN KEY (学生番号)
        REFERENCES 学生
        ON DELETE CASCADE
        ON UPDATE NO ACTION,
    FOREIGN KEY (科目番号)
        REFERENCES 科目
        ON DELETE CASCADE
        ON UPDATE NO ACTION)
```

行が削除・更新
された際の
動作を指定

外部キーの指定

制約

- 制約を問合せで表現可能
- 制約の名づけ

```
CREATE TABLE Sailors
  ( sid INTEGER,
    sname CHAR(10),
    rating INTEGER,
    age REAL,
    PRIMARY KEY (sid),
    CHECK ( rating >= 1
            AND rating <= 10 )
```

```
CREATE TABLE Reserves
  ( sname CHAR(10),
    bid INTEGER,
    day DATE,
    PRIMARY KEY (bid,day),
    CONSTRAINT noInterlakeRes
    CHECK ('Interlake' <>
           ( SELECT B.bname
             FROM Boats B
             WHERE B.bid=bid)))
```

更新操作(挿入)

「学生 S6(学生名‘川原’, 都市‘大阪’, 年令‘23’)
を関係“学生”に挿入せよ」

```
INSERT INTO 学生  
VALUES ('S6', N'川原', N'大阪', 23)
```

学生

学生番号	学生名	都市	年令
S1	山田	京都	19
S2	鈴木	大阪	20
S3	小島	奈良	22
S4	武田	京都	18
S5	高木	神戸	21



学生

学生番号	学生名	都市	年令
S1	山田	京都	19
S2	鈴木	大阪	20
S3	小島	奈良	22
S4	武田	京都	18
S5	高木	神戸	21
S6	川原	大阪	23

更新操作 (削除)

「科目J1 を削除せよ」

```
DELETE FROM 科目  
WHERE 科目番号 = 'J1'
```

科目

科目番号	科目名	先生	単位数
J1	データベース	田中	4
J2	計算理論	佐藤	2
J3	ハードウェア	小林	6
J4	データベース	大野	4
J5	OS	斎藤	5
J6	人工知能	田中	3



科目

科目番号	科目名	先生	単位数
J2	計算理論	佐藤	2
J3	ハードウェア	小林	6
J4	データベース	大野	4
J5	OS	斎藤	5
J6	人工知能	田中	3

更新操作(変更)

「科目J3 の科目名をハードウェア論,
先生を富田に変更し,
単位数を2単位削減せよ」

```
UPDATE 科目  
SET 科目名 = N'ハードウェア論',  
    先生 = N'富田',  
    単位数 = 単位数 - 2  
WHERE 科目番号 = 'J3'
```

科目

科目番号	科目名	先生	単位数
J1	データベース	田中	4
J2	計算理論	佐藤	2
J3	ハードウェア	小林	6
J4	データベース	大野	4
J5	OS	斎藤	5
J6	人工知能	田中	3



科目

科目番号	科目名	先生	単位数
J1	データベース	田中	4
J2	計算理論	佐藤	2
J3	ハードウェア論	富田	4
J4	データベース	大野	4
J5	OS	斎藤	5
J6	人工知能	田中	3

関係インスタンス（課題）

Sailors

<u>sid</u>	sname	rating	age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Reserves

<u>sid</u>	<u>bid</u>	day
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Boats

<u>bid</u>	bname	color
101	Interlake	Blue
102	Interlake	Red
103	Clipper	Green
104	Marine	Red

基本問合せ

```
SELECT      [DISTINCT] target-list
FROM        relation-list
WHERE       qualification
```

- *relation-list* 関係名のリスト
- *target-list* ほしい属性のリスト
- *qualification* 条件.
- DISTINCT 重複を除去. デフォルトでは、除去しない!

問合せの実行

概念レベル処理 (conceptual evaluation strategy) :

- relation-listに指定した関係の直積(cross-product)を求める.
- qualificationsの条件を満たさないタプルを削除.
- target-listに指定していない属性を削除
- DISTINCTが指定されていれば、重複の行を削除.

効率悪い！

例

Q1

```
SELECT S.sname  
FROM   Sailors S, Reserves R  
WHERE  S.sid=R.sid AND R.bid=103
```

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

範囲変数(Range Variables)

- 二つ以上の関係を区別に用いる. 特に, 同じ関係が2回以上from節に現れる場合.

```
SELECT S.sname  
FROM   Sailors S, Reserves R  
WHERE  S.sid=R.sid AND bid=103
```

Q2

OR

```
SELECT sname  
FROM   Sailors, Reserves  
WHERE  Sailors.sid=Reserves.sid  
       AND bid=103
```

Q3

少なくとも一艘の*boat*を予約した*sailors*を求めよ

Q4

```
SELECT S.sid  
FROM Sailors S, Reserves R  
WHERE S.sid=R.sid
```

Q5

“DISTINCT”を追加した場合、違いがあるでしょうか？

Q6,Q7

Select節に *S.sid* ではなく、*S.sname* である場合、答えはどうなるのでしょうか？この場合、“DISTINCT”を追加したら違いがあるでしょうか？

少なくとも一艘のboatを予約したsailorsをratingの高い順で並べよ

Q8

```
SELECT S.sid  
FROM Sailors S, Reserves R  
WHERE S.sid=R.sid  
ORDER BY S.rating DESC
```

- 順序付き検索

Q9

小さい順に出力したい際には DESC の代わりに ASC を指定する

Expressions and Strings

Q10A

```
SELECT S.age, age1=S.age-5, 2*S.age AS age2  
FROM Sailors S  
WHERE S.sname LIKE 'B_%B'
```

- 数値式と文字列のパターンマッチング
- AS と = を利用して結果における属性名を定義.
- LIKE を文字列のマッチングに用いる.
 - _ : 任意の1文字
 - % : 長さ0 以上の任意文字.

Q10B

```
SELECT S.age, S.age-5 AS age1, 2*S.age AS age2  
FROM Sailors S  
WHERE S.sname LIKE 'B_%B'
```

赤か緑のboatを予約したsailorsのsidは？

Q11A

```
SELECT S.sid  
FROM Sailors S, Boats B, Reserves R  
WHERE S.sid=R.sid AND R.bid=B.bid  
AND (B.color='Red' OR B.color='Green')
```

OR を AND にしたらどうなるのでしょうか？

Q11B

- UNION: 集合和
- EXCEPT(集合差) や
INTERSECT(共通集合) もある

```
SELECT S.sid  
FROM Sailors S, Boats B, Reserves R  
WHERE S.sid=R.sid AND R.bid=B.bid  
AND B.color='Red'  
UNION  
SELECT S.sid  
FROM Sailors S, Boats B, Reserves R  
WHERE S.sid=R.sid AND R.bid=B.bid  
AND B.color='Green'
```

赤と緑のboatを予約したsailorsのsidは？

Q12A

```
SELECT S.sid  
FROM Sailors S, Boats B1, Reserves R1,  
      Boats B2, Reserves R2  
WHERE S.sid=R1.sid AND R1.bid=B1.bid  
      AND S.sid=R2.sid AND R2.bid=B2.bid  
      AND (B1.color='Red' AND B2.color='Green')
```

Q12B

- **INTERSECT:** 共通集合
- SQL/92 からサポート

SELECT S.sid Key field!
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.bid=B.bid
 AND B.color='Red'
INTERSECT
SELECT S.sid
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.bid=B.bid
 AND B.color='Green'

副問合せ

- SQLではある問合せの中に別の問合せを含むことができる

Find names of sailors who've reserved boat #103:

Q13 SELECT S.sname
 FROM Sailors S
 WHERE S.sid IN (SELECT R.sid
 FROM Reserves R
 WHERE R.bid=103)

副問合せ

Find names of sailors who've reserved boat #103:

```
Q14  SELECT S.sname  
      FROM Sailors S  
      WHERE EXISTS (SELECT *  
                     FROM Reserves R  
                     WHERE R.bid=103 AND S.sid=R.sid)
```

Find sailors with at most one reservation for boat #103

```
Q15  SELECT S.sname  
      FROM Sailors S  
      WHERE UNIQUE (SELECT R.bid  
                     FROM Reserves R  
                     WHERE R.bid=103 AND S.sid=R.sid)
```

その他の集合演算

- IN, EXISTS と UNIQUE のほかに , NOT IN, NOT EXISTS や NOT UNIQUEもある.
- $op\ ANY$, $op\ ALL$, $op\ IN$ op: $>$, $<$, $=$, \geq , \leq , \neq
- 例: *Find sailors whose rating is greater than that of some sailor called Horatio:*

```
Q16  SELECT *
      FROM Sailors S
      WHERE S.rating > ANY (SELECT S2.rating
                             FROM Sailors S2
                             WHERE S2.sname='Horatio')
```

INTERSECT を INで書き換え

赤と緑のboatを予約したsailors のsidは？ (P18)

```
Q17 SELECT S.sid
  FROM Sailors S, Boats B, Reserves R
 WHERE S.sid=R.sid AND R.bid=B.bid AND B.color='Red'
       AND S.sid IN (SELECT S2.sid
                      FROM Sailors S2, Boats B2, Reserves R2
                     WHERE S2.sid=R2.sid AND R2.bid=B2.bid
                           AND B2.color='Green')
```

- 同様に, EXCEPT をNOT INで書き換えられる.

Q18

商

すべてのboatsを予約したsailorsは？

(1)

```
SELECT S.sname  
FROM Sailors S  
WHERE NOT EXISTS  
(SELECT B.bid  
FROM Boats B)  
EXCEPT  
(SELECT R.bid  
FROM Reserves R  
WHERE R.sid=S.sid)
```

Q19

(2) SELECT S.sname
FROM Sailors S

WHERE NOT EXISTS (SELECT B.bid

Sailors S such that ...

FROM Boats B

WHERE NOT EXISTS (SELECT R.bid

there is no boat B without ...

FROM Reserves R

WHERE R.bid=B.bid

a Reserves tuple showing S reserved B

AND R.sid=S.sid))

集約

- SELECT句には、平均、最大、最小、合計などの集合関数を指定
- 関係代数、関係論理には無い機能

Q20

```
SELECT COUNT (*)
FROM Sailors S
```

Q21

```
SELECT AVG (S.age)
FROM Sailors S
WHERE S.rating=10
```

Q22

```
SELECT S.sname
FROM Sailors S
WHERE S.rating= (SELECT MAX(S2.rating)
                  FROM Sailors S2)
```

Q23

```
SELECT AVG ( DISTINCT S.age)
FROM Sailors S
WHERE S.rating=10
```

Q24

```
SELECT COUNT (DISTINCT S.rating)
FROM Sailors S
WHERE S.sname='Bob'
```

COUNT (*)
COUNT ([DISTINCT] A)
SUM ([DISTINCT] A)
AVG ([DISTINCT] A)
MAX (A)
MIN (A)

単一カラム

最高齢のsailor(s)の名前と年齢を？

Q25 (1) SELECT S.sname, MAX (S.age)
 FROM Sailors S

Q26 (2) SELECT S.sname, S.age
 FROM Sailors S
 WHERE S.age =
 (SELECT MAX (S2.age)
 FROM Sailors S2)

Q27 (3) SELECT S.sname, S.age
 FROM Sailors S
 WHERE (SELECT MAX (S2.age)
 FROM Sailors S2)
 = S.age

グループ

- ・グループ分けて情報を集約したい, 分析したい
- ・例: それぞれのratingレベルにおける最年少の*sailor*を知りたい
 - Rating levelの数を知らないし, ratingの値も知らない.
 - Ratingの値が1-10であることを知っていれば:

For $i = 1, 2, \dots, 10$:

```
SELECT MIN (S.age)
FROM Sailors S
WHERE S.rating = i
```

GROUP BY と HAVING

```
SELECT      [DISTINCT] target-list
FROM        relation-list
WHERE       qualification
GROUP BY    grouping-list
HAVING     group-qualification
```

target-list : 属性集合 と 集計演算 (e.g., MIN (*S.age*))から構成される.

Target-listにある属性集合は*grouping-list*のサブセットである.

グループは、 *grouping-list*にある属性が同じ値を持つタプルの集合である.

概念レベルの処理

relation-list にある関係の直積を求める

qualification を満たさないタプルを除去；必要な属性も削除；

grouping-list の属性を用いてタプルをグルーピング

group-qualification を用いて条件を満たさないグループを除去.

グループごとに答えのタプルを出力.

Find age of the *youngest* sailor with $age \geq 18$,
for each rating with at least 2 such sailors

Q28

```
SELECT S.rating, MIN (S.age)
      AS minage
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating
HAVING COUNT (*) > 1
```

Answer relation:

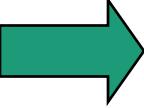
rating	minage
3	25.5
7	35.0
8	25.5

Sailors instance:

sid	sname	rating	age
22	dustin	7	45.0
29	brutus	1	33.0
31	lubber	8	55.5
32	andy	8	25.5
58	rusty	10	35.0
64	horatio	7	35.0
71	zorba	10	16.0
74	horatio	9	35.0
85	art	3	25.5
95	bob	3	63.5
96	frodo	3	25.5

*Find age of the youngest sailor with age ≥ 18 ,
for each rating with at least 2 such sailors.*

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5



rating	age
1	33.0
3	25.5
3	63.5
3	25.5
7	45.0
7	35.0
8	55.5
8	25.5
9	35.0
10	35.0

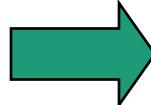


rating	minage
3	25.5
7	35.0
8	25.5

Find age of the youngest sailor with age ≥ 18 , for each rating with at least 2 such sailors and with every sailor under 60.

HAVING COUNT (*) > 1 AND EVERY (S.age <=60)

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5



rating	age
1	33.0
3	25.5
3	63.5
3	25.5
7	45.0
7	35.0
8	55.5
8	25.5
9	35.0
10	35.0



rating	minage
7	35.0
8	25.5

EVERYではなくANYだったら？

*Find age of the youngest sailor with age ≥ 18 , for each rating with at least 2 sailors *between 18 and 60*.*

Q29

```
SELECT S.rating, MIN (S.age)
      AS minage
FROM Sailors S
WHERE S.age >= 18 AND S.age <= 60
GROUP BY S.rating
HAVING COUNT (*) > 1
```

Answer relation:

rating	minage
3	25.5
7	35.0
8	25.5

Sailors instance:

sid	sname	rating	age
22	dustin	7	45.0
29	brutus	1	33.0
31	lubber	8	55.5
32	andy	8	25.5
58	rusty	10	35.0
64	horatio	7	35.0
71	zorba	10	16.0
74	horatio	9	35.0
85	art	3	25.5
95	bob	3	63.5
96	frodo	3	25.5

For each red boat, find the number of reservations for this boat

Q30A

```
SELECT B.bid, COUNT(*) AS scount  
FROM Boats B, Reserves R  
WHERE R.bid=B.bid AND B.color='red'  
GROUP BY B.bid
```

Q30B

- *B.color='red'* を WHERE 節から削除して, HAVINGでこの条件を表現できるか？

*Find age of the youngest sailor with age > 18,
for each rating with at least 2 sailors (of any age)*

Q31A

```
SELECT S.rating, MIN (S.age)
FROM Sailors S
WHERE S.age > 18
GROUP BY S.rating
HAVING 1 < (SELECT COUNT (*)
               FROM Sailors S2
              WHERE S.rating=S2.rating)
```

- HAVING節における副問合せの利用
- HAVING節は以下の場合はどうなるか？
 - HAVING COUNT(*) >1

Q31B

結合

- CROSS JOIN
- NATURAL JOIN
- INNER JOIN
- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN
- UNION JOIN

SQL課題

- 9ページに示しているデータベースを作成してください。テーブルを作成するためのSQL文、データ追加のSQL文（各テーブルの最初のレコードを挿入するものでよい）を示してください。
- 10ページ以降の問合せ(Q2-Q31B)を実行し、その結果を確認する。SQL文とその結果を報告すること。実行できない問合せや正しくない問合せがあれば、その理由について説明してください。

SQLite (<http://www.sqlite.org/>)

- 手軽に利用できる関係データベースシステム
 - 「Download」から環境に合わせて「command-line shell」を取得

“sample.db”はデータベース名。

(ここで指定した名前でファイルが作成される。

既にファイルがある場合はそこから読み込む。)

```
> sqlite3 sample.db
```

```
> ...
```

```
sqlite> create table 学生 (学生番号, 学生名, 都市, 年令);
```

```
sqlite> insert into 学生 values('S1', '山田', '京都', 19);
```

```
sqlite> insert into 学生 values('S2', '鈴木', '大阪', 20);
```

```
sqlite> ...
```

```
sqlite> select * from 学生;
```

テーブルの作成。

一般的には各属性に対して型指定が必要だが、
SQLiteでは省略できる。

文の終わりを示すために
最後に「;」をつける。

SQLiteでは N'鈴木' の
ようにするとエラーになる 全角半角に注意！

(区切りのコンマは半角)

SQLite コマンド

- 「.」から始まるのは
SQLite独自コマンド

コマンド	説明
.help	SQLite独自のコマンドの説明表示
.tables	表の一覧の表示
.schema 表名	表のスキーマ(定義)を表示
.output ファイル名	以降の結果をファイルに出力 例: .output sqliteresult.txt
.output stdout	以降の結果を画面に表示するモードに戻る
.read ファイル名	ファイルに書かれたSQLを実行
.exit	sqliteを終了

SQLite 実行までの注意点

- 環境にあったものを用いる
 - WindowsだったらWindows版、 MacだったらMac版、 LinuxだったらLinux版
- Linuxでの実行には ./ をつける
(SQLiteの注意点ではなくLinuxの注意点)
 - sqlite3 があるディレクトリをカレントディレクトリにして

```
$ ./sqlite3 sample.db
```

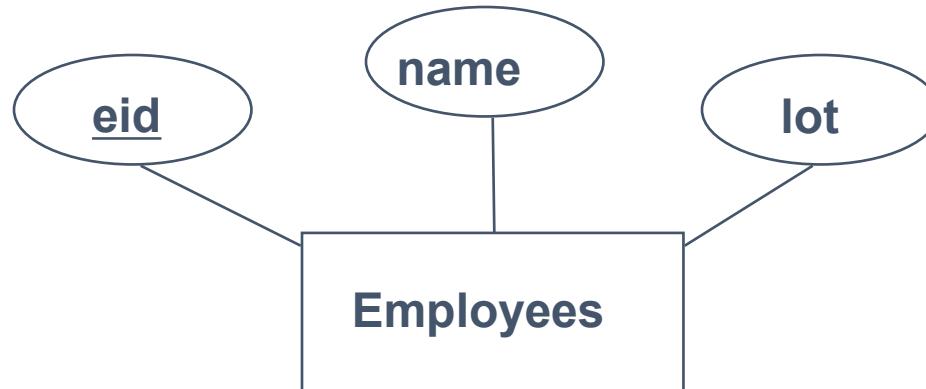
データベース： 実体関連モデル

馬

実体関連モデル (Entity-Relationship Model)

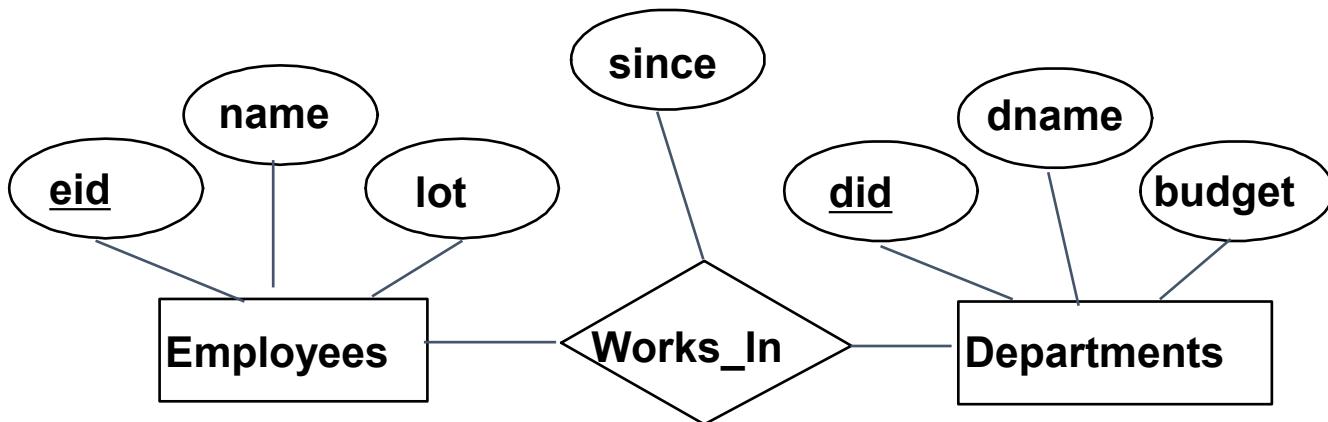
- ERモデル
- 実世界のデータを
 - 実体 (entity)
 - 関連 (relationship)
 - 属性 (attribute)
- で表現
- データベースの概念設計のためには良く利用

実体, 実体集合, 属性



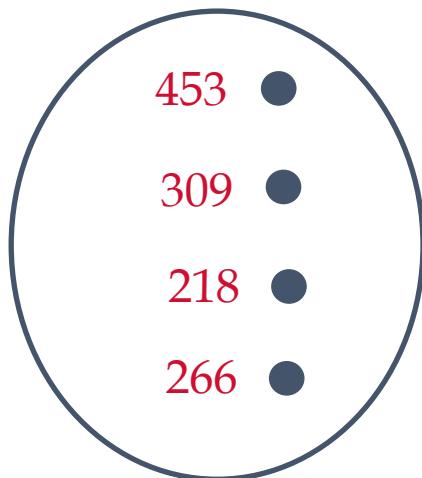
- 実体 (entity) ... 実世界において識別可能な物体や事象
- 実体集合 (entity set) ... 同様の実体の集合
- 属性 (attribute)
 - 定義域 (domain)
例：属性nameの定義域は20文字以内の文字列
- キー (key) ... 実体を一意に識別する属性の極小集合
 - 候補キー (candidate key), 主キー (primary key)

関連, 関連集合

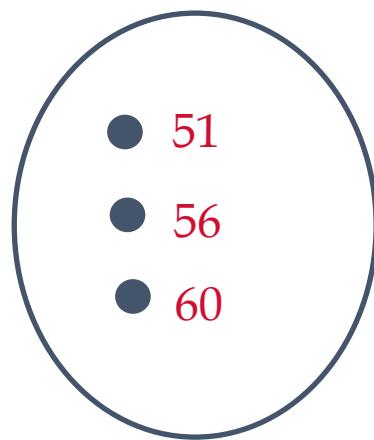


- 関連 (relationship) ... 複数個の実体間の対応
- 関連集合 (relationship set) ... 同様の関連の集まり
- 記述的属性 (descriptive attribute)
- 関連は参加している実体によって唯一に識別される
{eid, did} がWorks_Inのキー

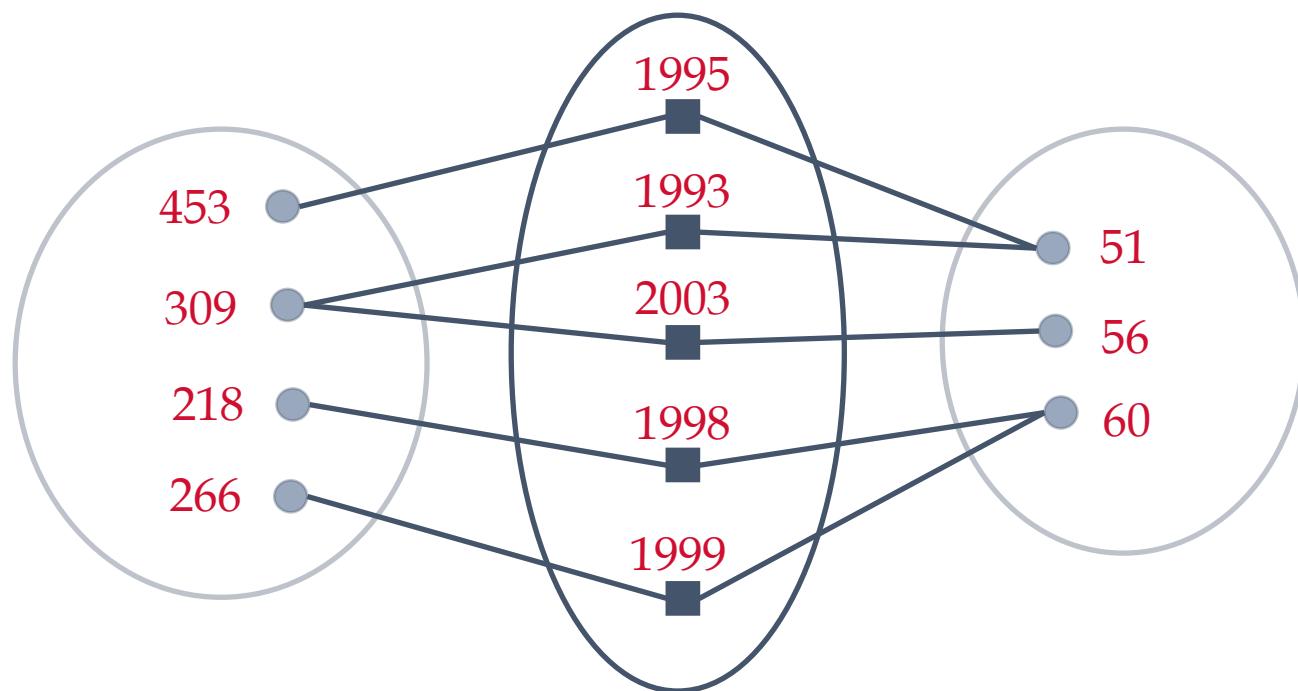
Employees実体集合のインスタンス例



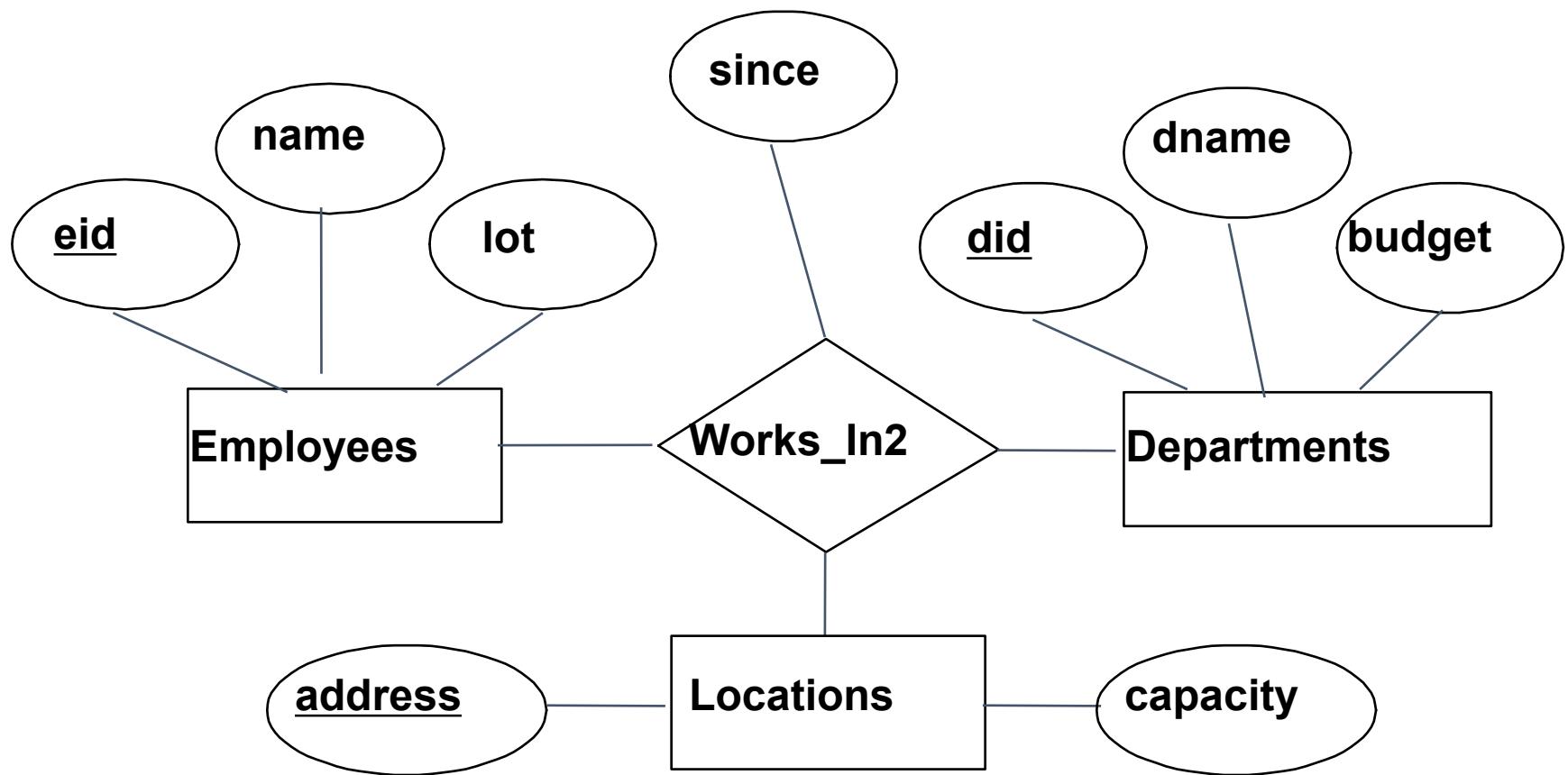
Departments実体集合のインスタンス例



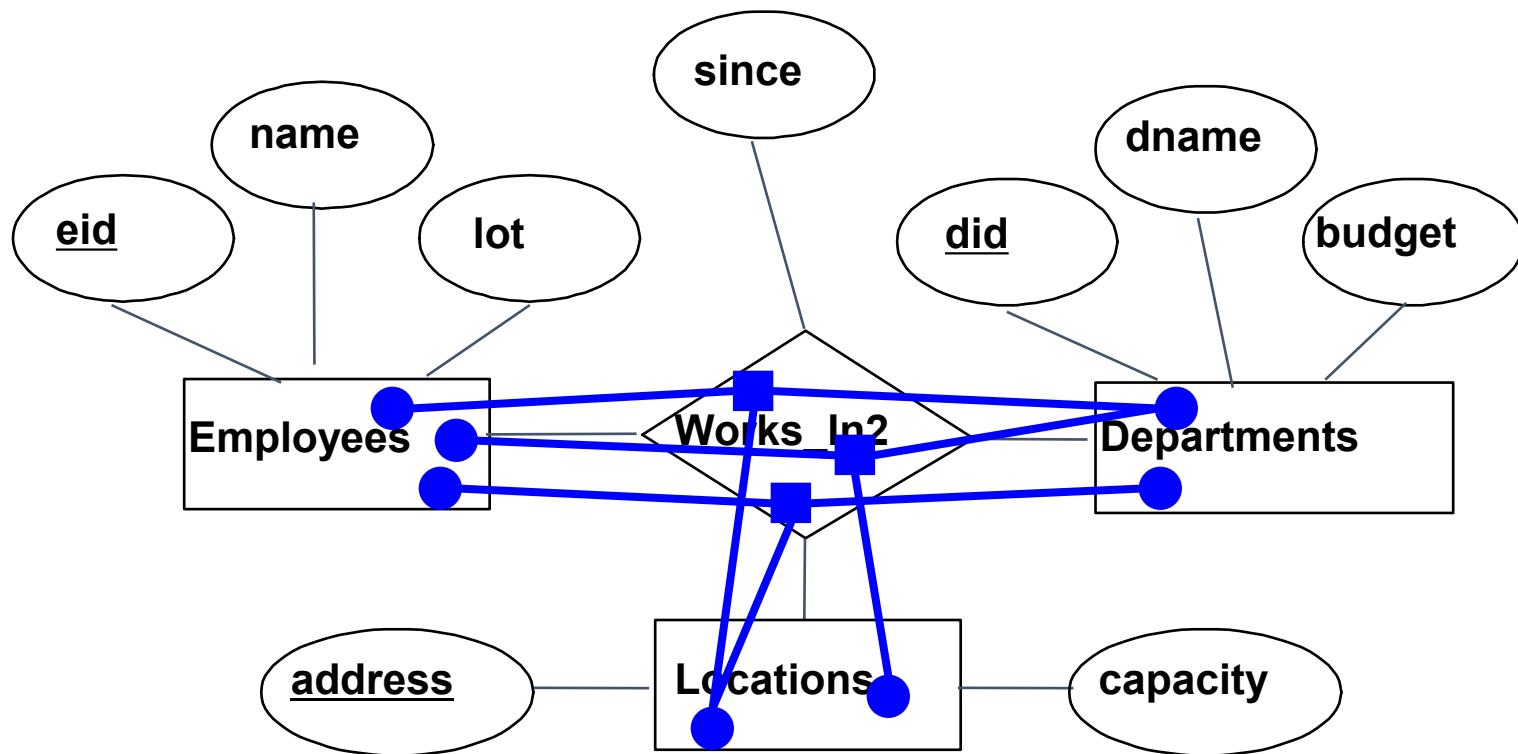
Works_In関連集合のインスタンス例 I₁



三項関連集合

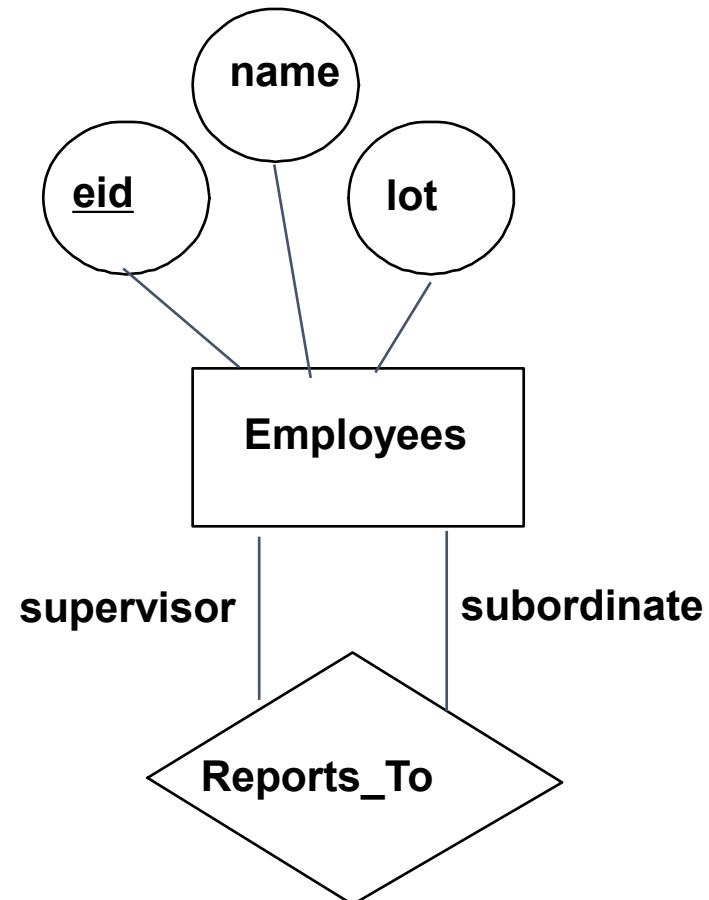


三項関連集合



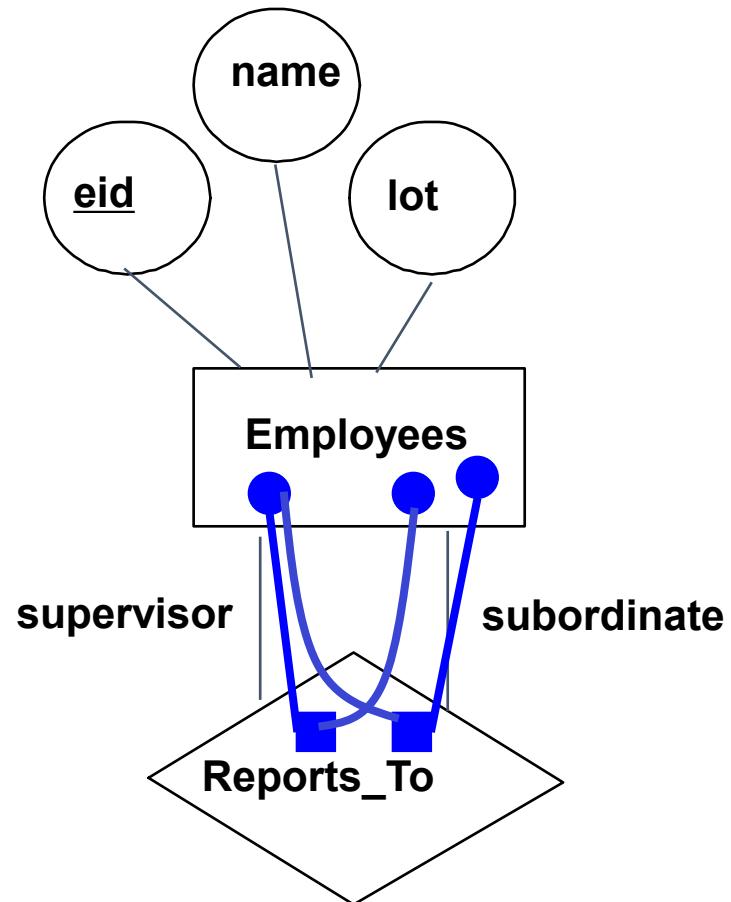
同一の実体集合間の関連集合

- 役割 (role) を指定する
必要がある。
 - role indicator

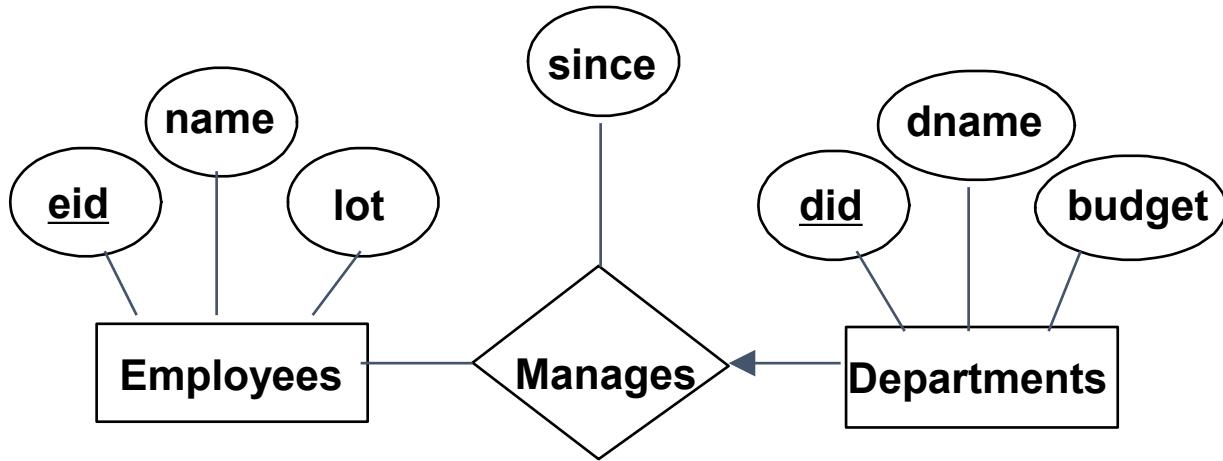


同一の実体集合間の関連集合

- 役割 (role) を指定する
必要がある。
 - role indicator

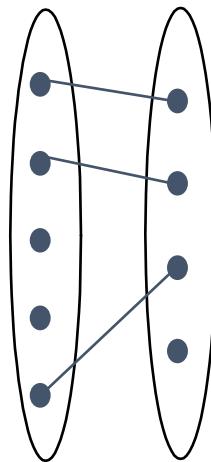


キー制約 (Key Constraints)

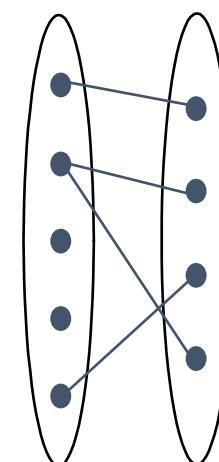


- ある **Departments** 実体が与えられれば、それが現れる **Manages** 関連を唯一に決定できる。
(ある一つの **department** は多くとも一人の **employee** が **manage** する。また、ある一人の **employee** は一つ以上の **department** を **manage** しても良い)
- {did} が **Manages** のキー

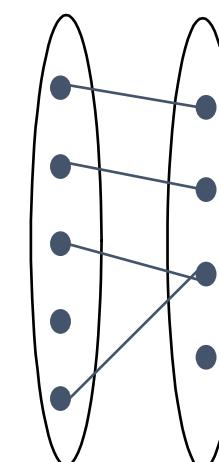
二項関連(binary relationship)の種類



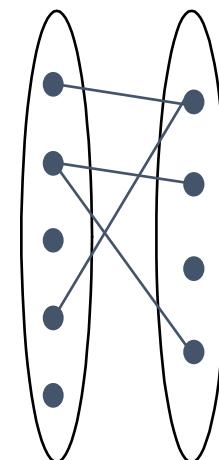
1-to-1



1-to Many

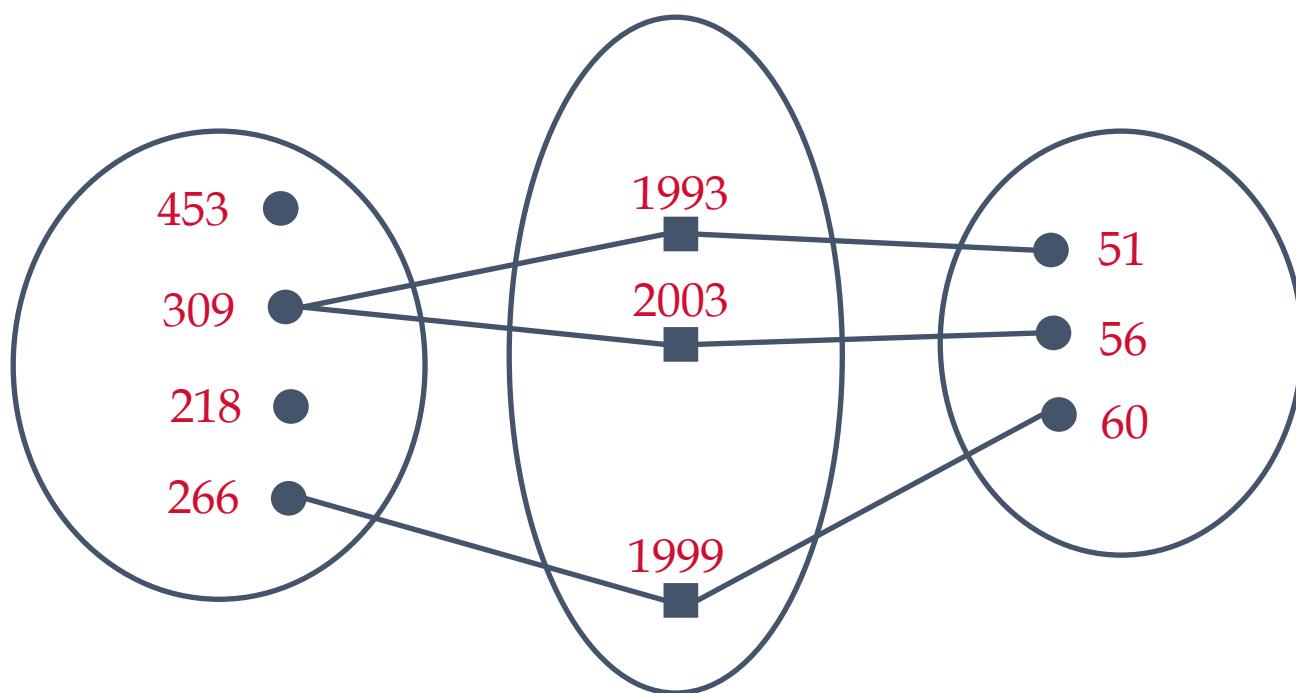


Many-to-1



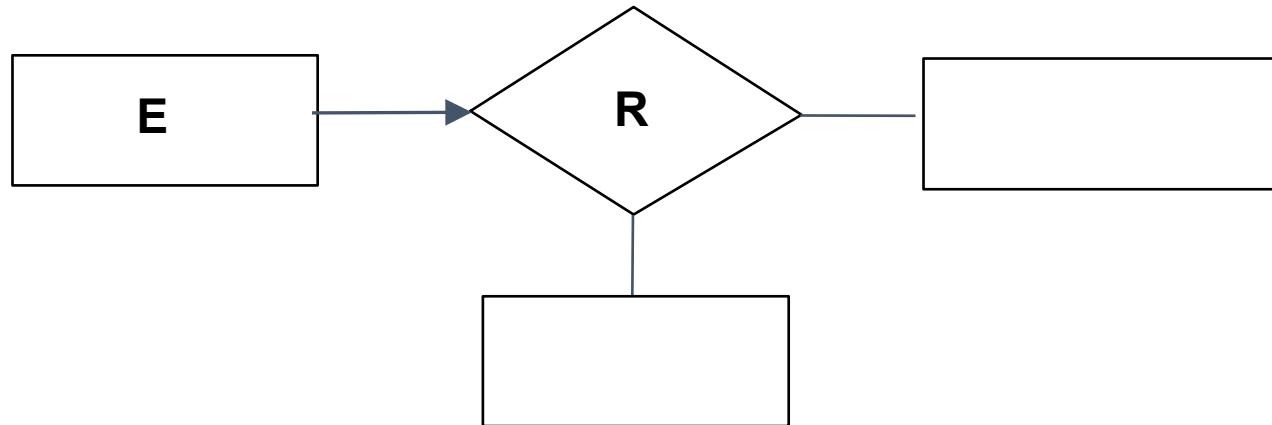
Many-to-Many

Manages関連集合のインスタンス例 |₂

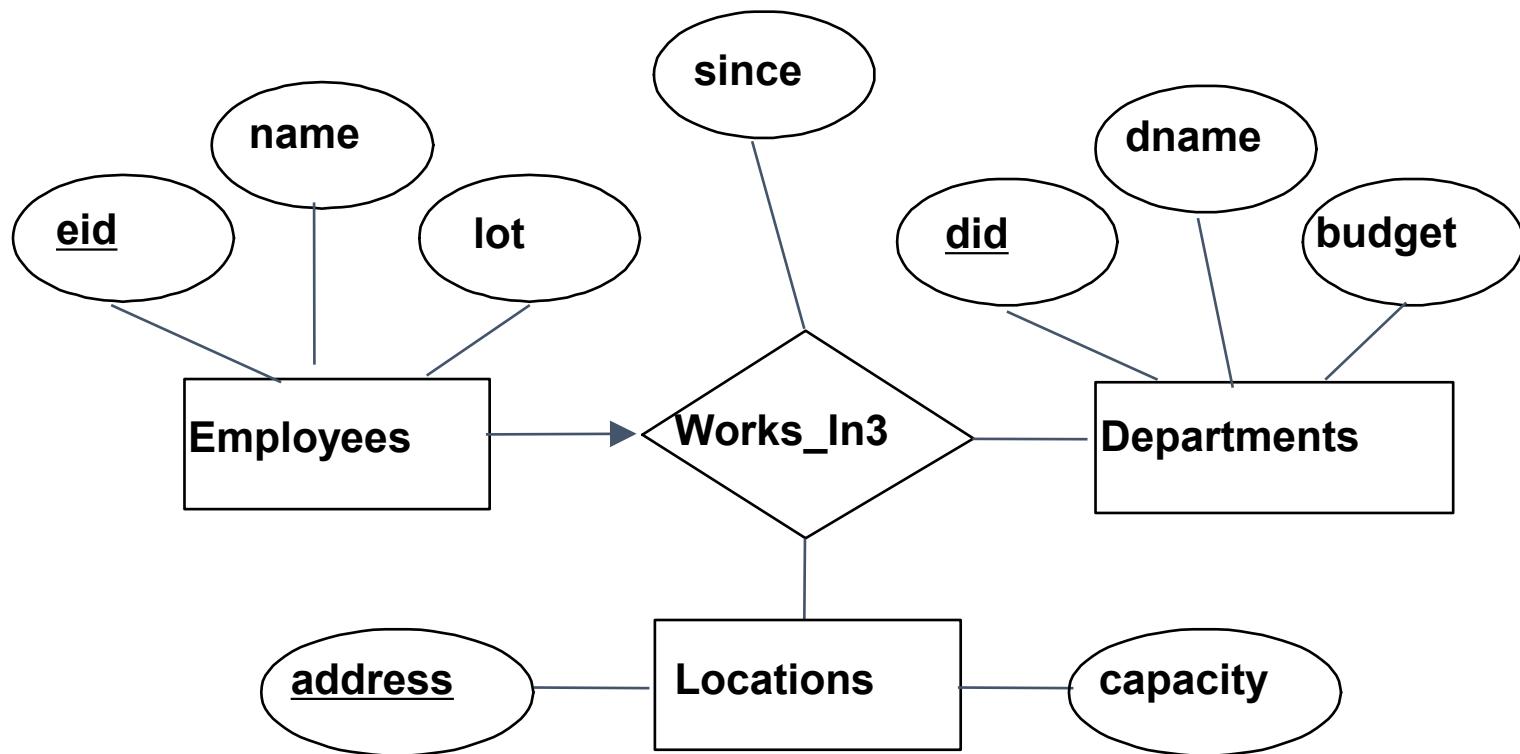


キー制約を持つ三項関連集合

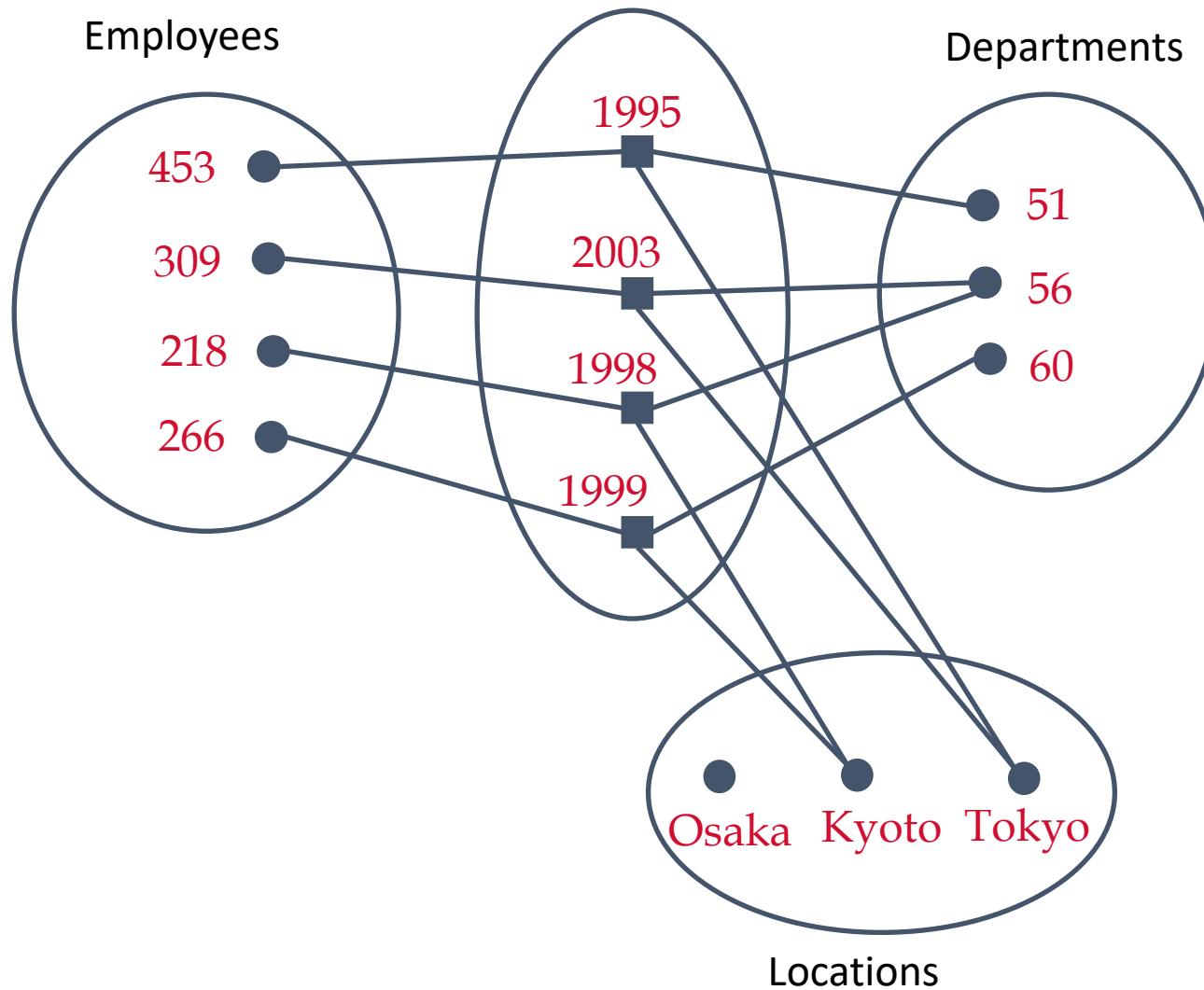
実体集合Eから関連集合Rへのキー制約が成立するということは、Eの実体はRの高々一つの関連にのみ現れるということ。



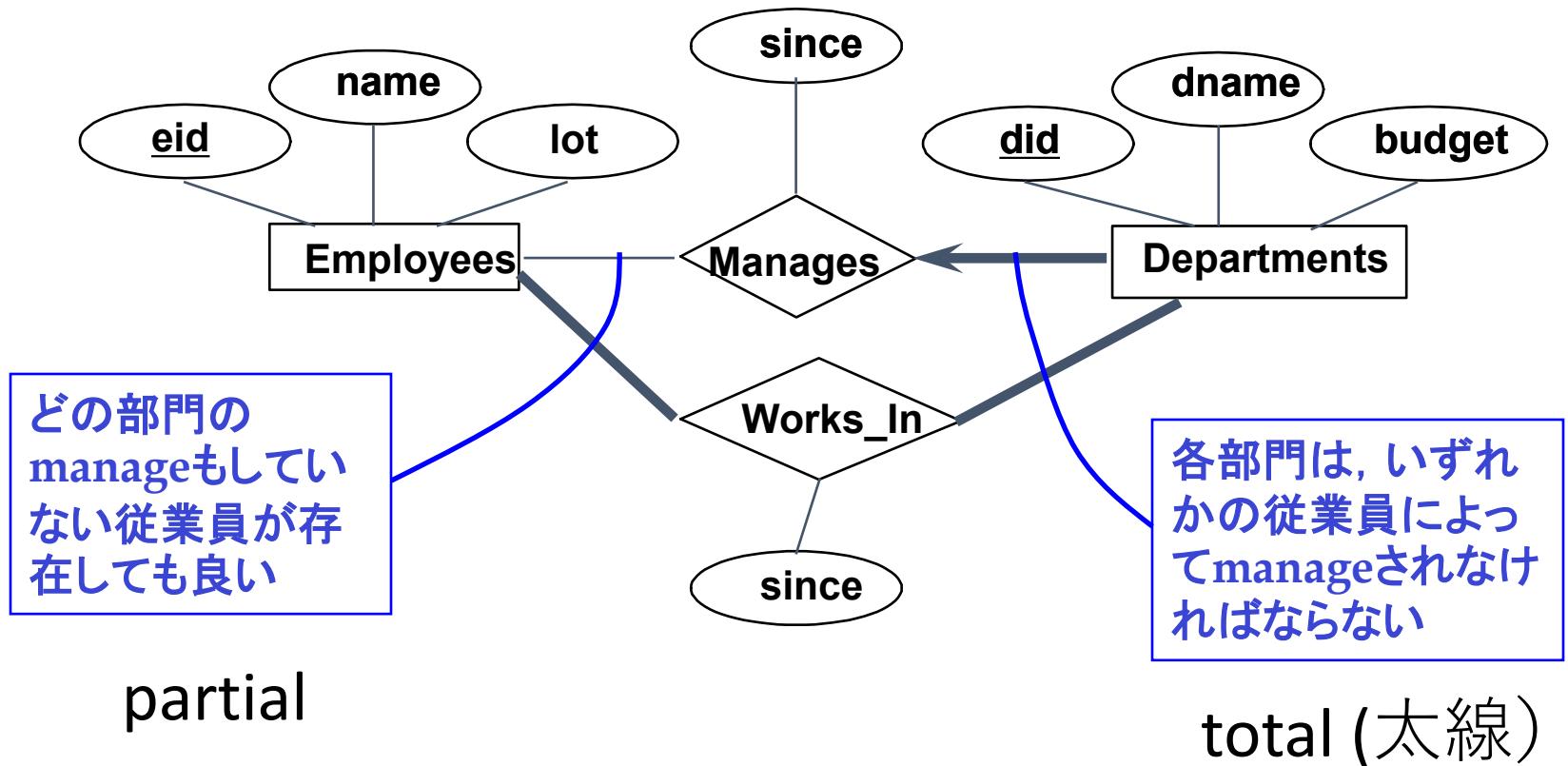
キー制約を持つ三項関連集合の例



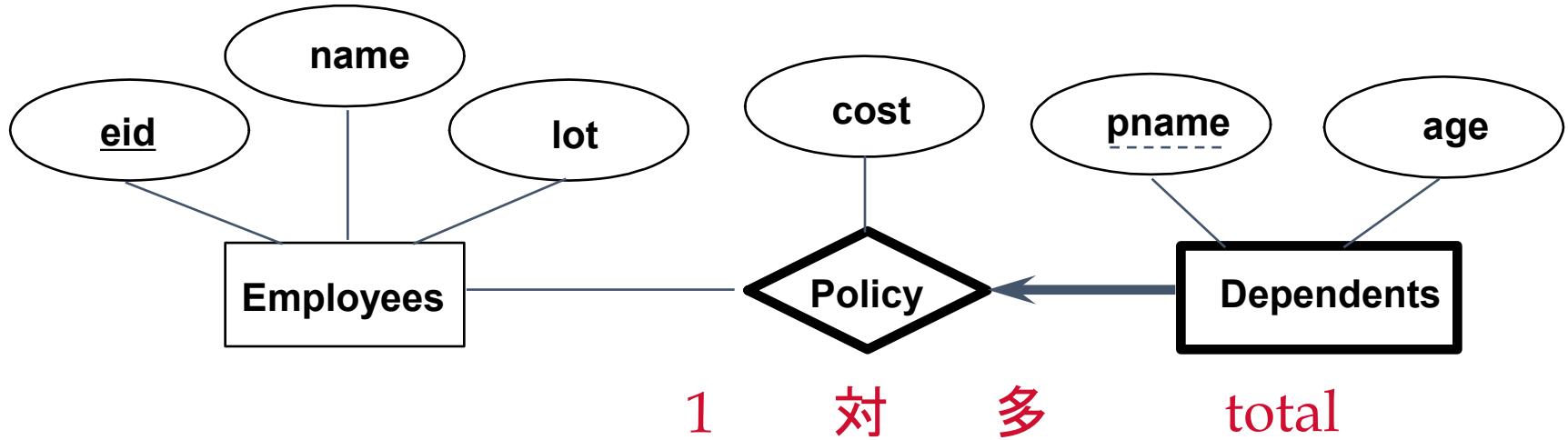
Works_In3関連集合のインスタンス例



参加制約 (Participation Constraints)



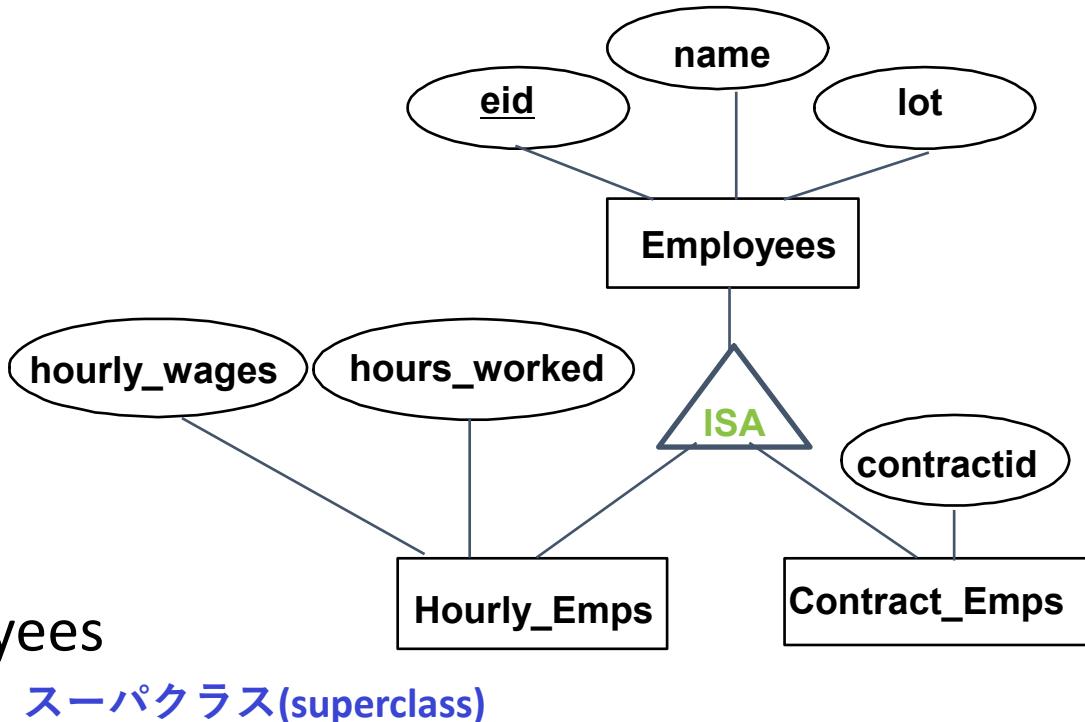
弱実体(Weak Entities)



- “Dependents”は弱実体集合
- 弱実体は、部分キー (= 自身の属性のいくつか) と **identifying owner** と呼ばれる別の実体の主キーを合わせてはじめて唯一に識別可能

クラス階層

- 実体集合Employeesの属性は、実体集合Hourly_Empsによって継承される(inherited).



- Hourly_Emps ISA Employees

部分クラス(subclass)

スーパーカラス(superclass)

- 二つの見方

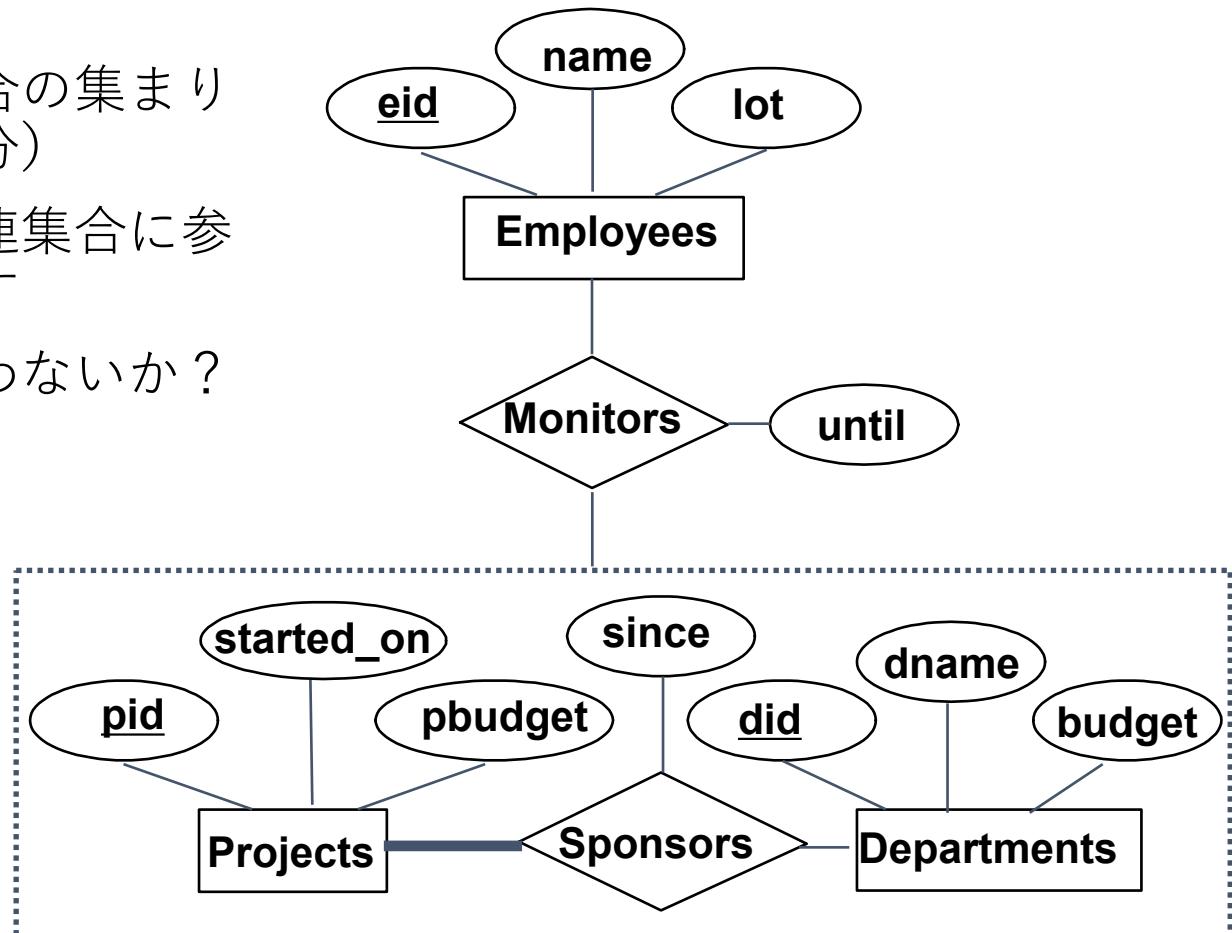
- Employeesは、二つの部分クラスに特殊化される(specialized)
- Hourly_EmpsとContract_Empsは、Employeesによって汎化される(generalized)

クラス階層

- クラス階層に関する二種類の制約
 - overlapping constraints
二つの部分クラスが同じ実体を含んで良いかどうかを決定
 - 何も指定がない場合は含まないと仮定
 - 含む場合は,
`Contract_Emps OVERLAPS Senior_Emps` のように記述
 - covering constraints
部分クラスすべての実体によってスーパークラスのすべての実体を包含するかどうかを決定
汎化階層の場合はcovering constraintsが成立
 - 何も指定がない場合は成立しないと仮定
 - 成立する場合は,
`Motorboats AND Cars COVER Motor_Vehicle`
のように記述

集約(Aggregation)

- 実体集合と関連集合の集まり
(点線で囲った部分)
- 関連集合が他の関連集合に参加することを表わす
- なぜ三項関連を使わないか？



ER図作成時の選択肢

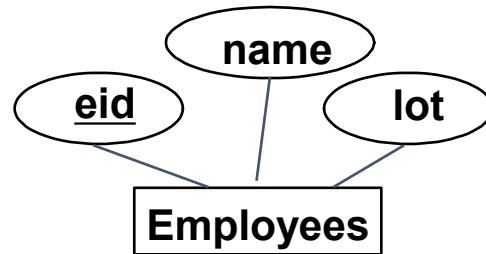
実体 v.s. 属性

実体 v.s. 関連

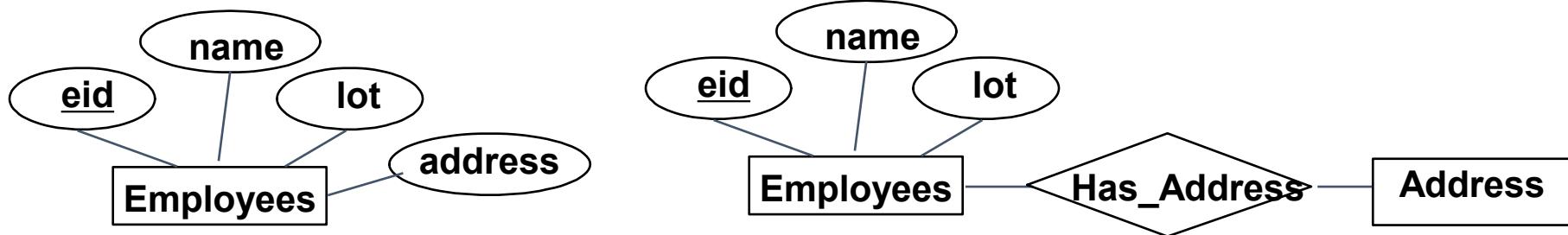
二項関連 v.s. 三項関連 v.s. ... v.s. n項関連

集約？

Entity vs. Attribute

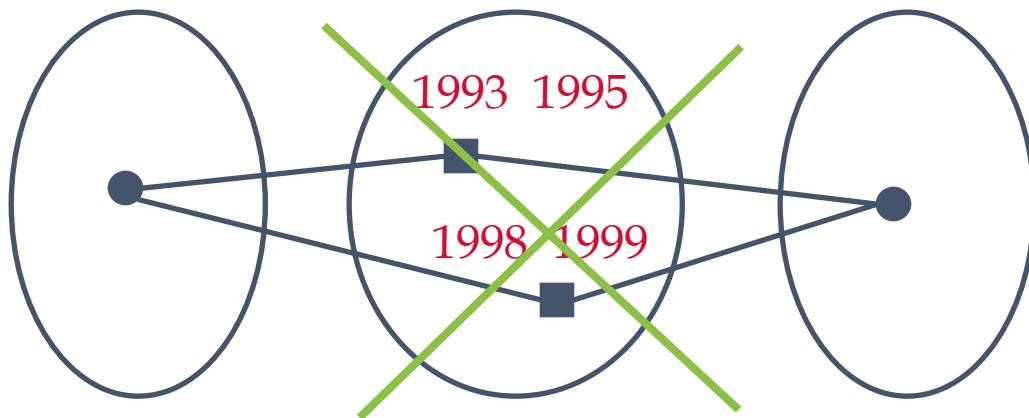
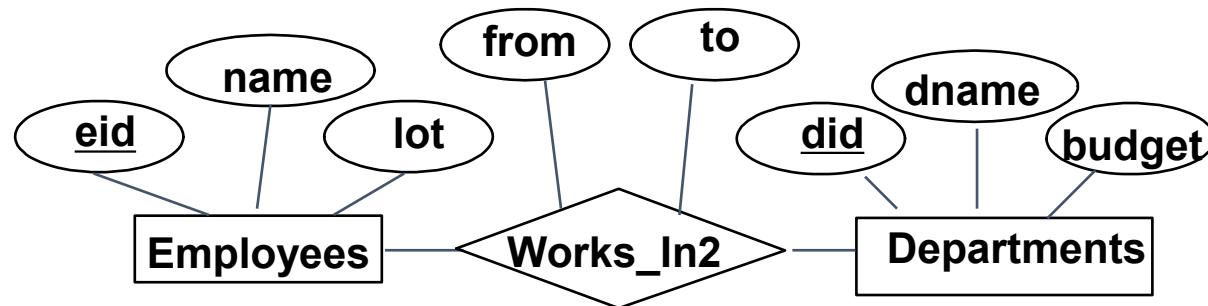


住所情報の追加

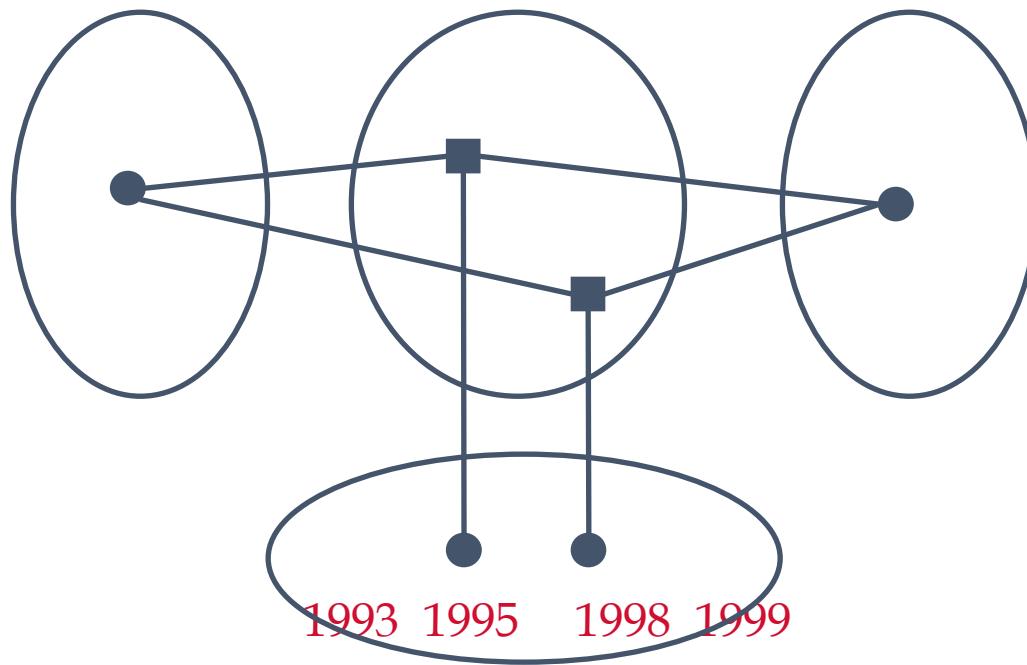
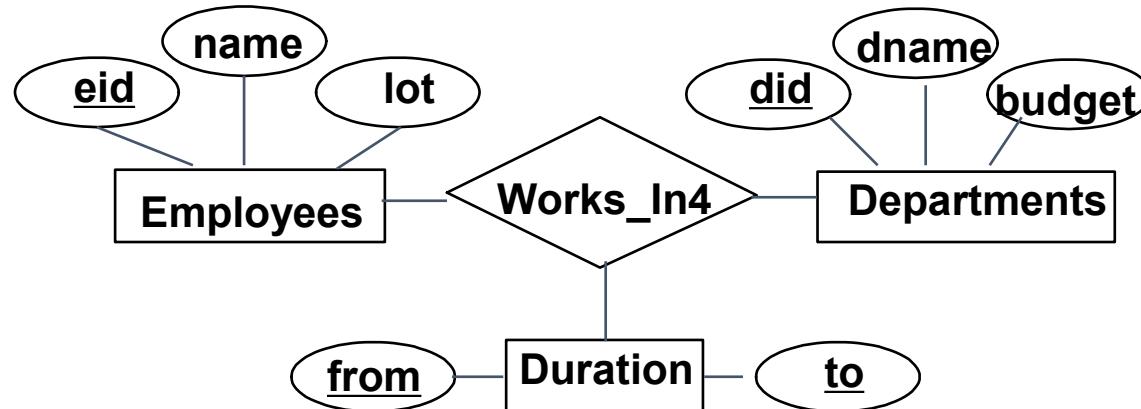


Entity vs. Attribute

- 一人の従業員が一つの部門で二つ以上の期間働くことができる場合

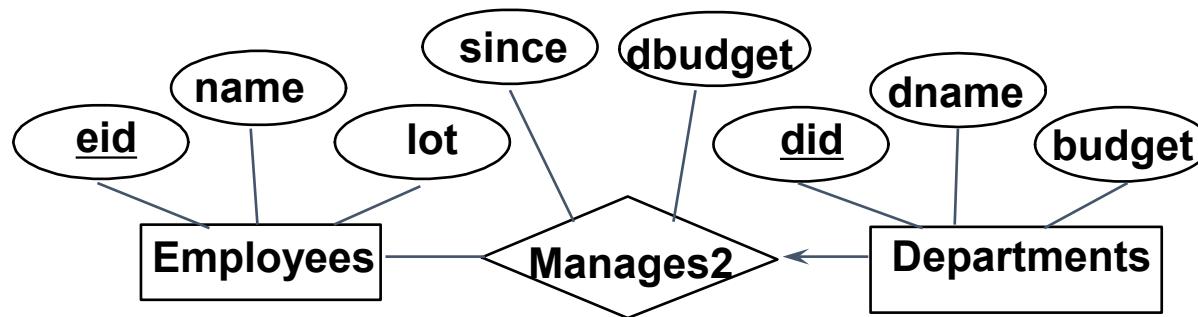


Entity vs. Attribute



Entity vs. Relationship

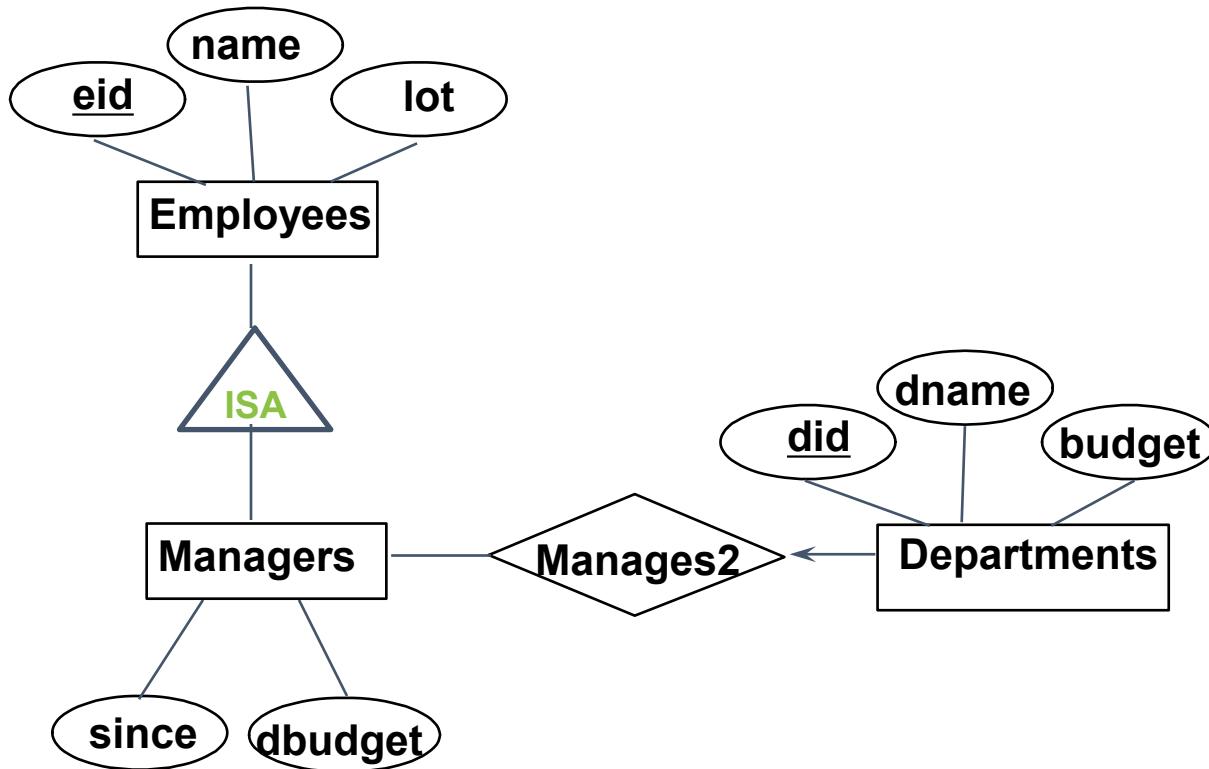
- manager が自分が管理する部門ごとに裁量予算(dbudget)を持っているなら以下のER図で良い.



- しかし、裁量予算(dbudget)とは、manager が管理するすべての部門に対するものの合計であったとすると
 - dbudget の値は manager が管理する部門ごとに繰り返され冗長となる.

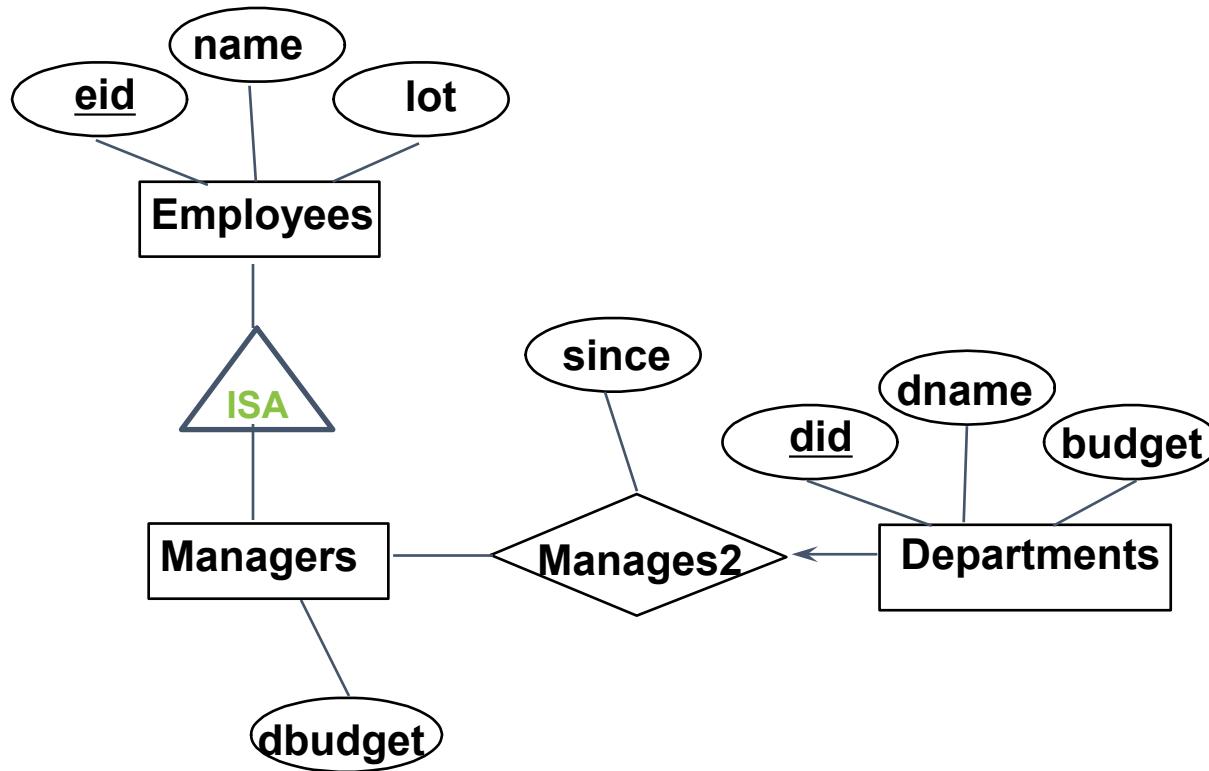
解決法 – case 1

- 従業員が複数の部門をmanageするとき，すべての部門で同じ日からmanageする場合



解決法 – case 2

- 従業員が複数の部門をmanageするとき， 部門によって manageを開始する日が異なる場合



ER図作成時の選択肢

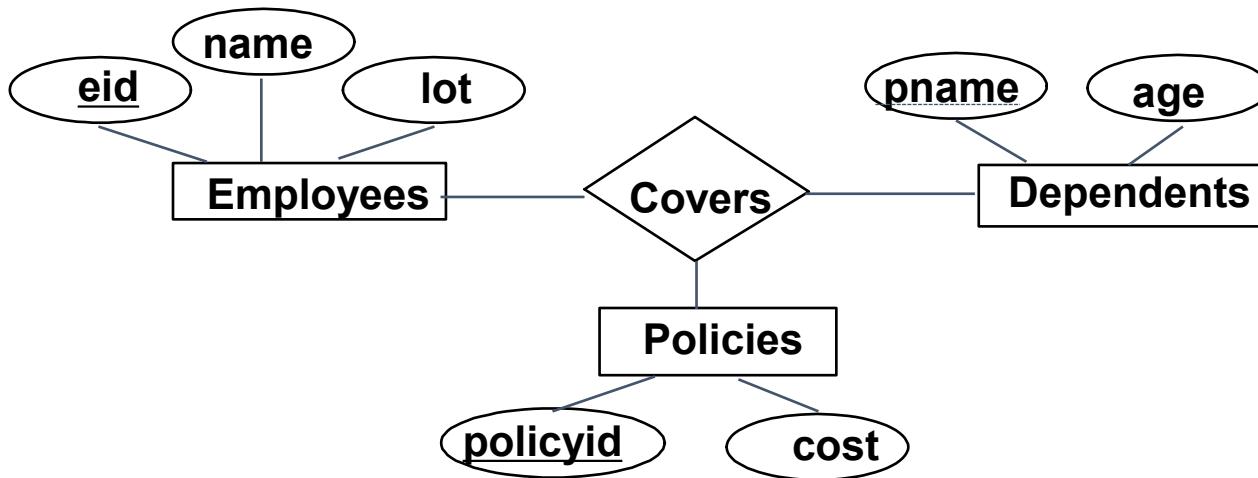
実体 v.s. 属性

実体 v.s. 関連

二項関連 v.s. 三項関連 v.s. ... v.s. n項関連

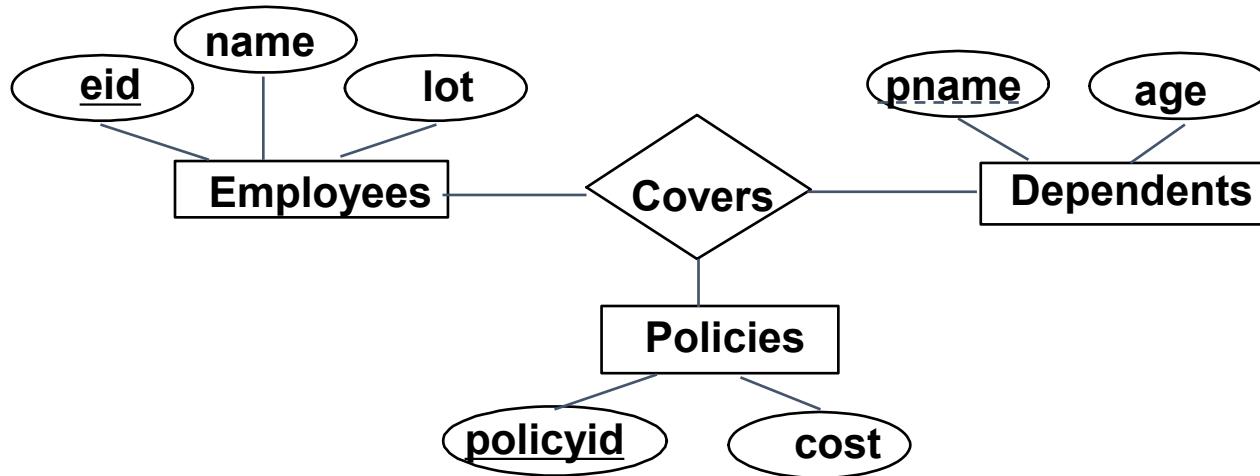
集約？

二項関連 vs. 三項関連



- 一人の従業員 (employee)が複数の保険(policy)を持つ
- 一つの保険は何人かの従業員に所有
- 一人の扶養家族(dependent)は複数の保険にカバー

二項関連 vs. 三項関連(Cont.)

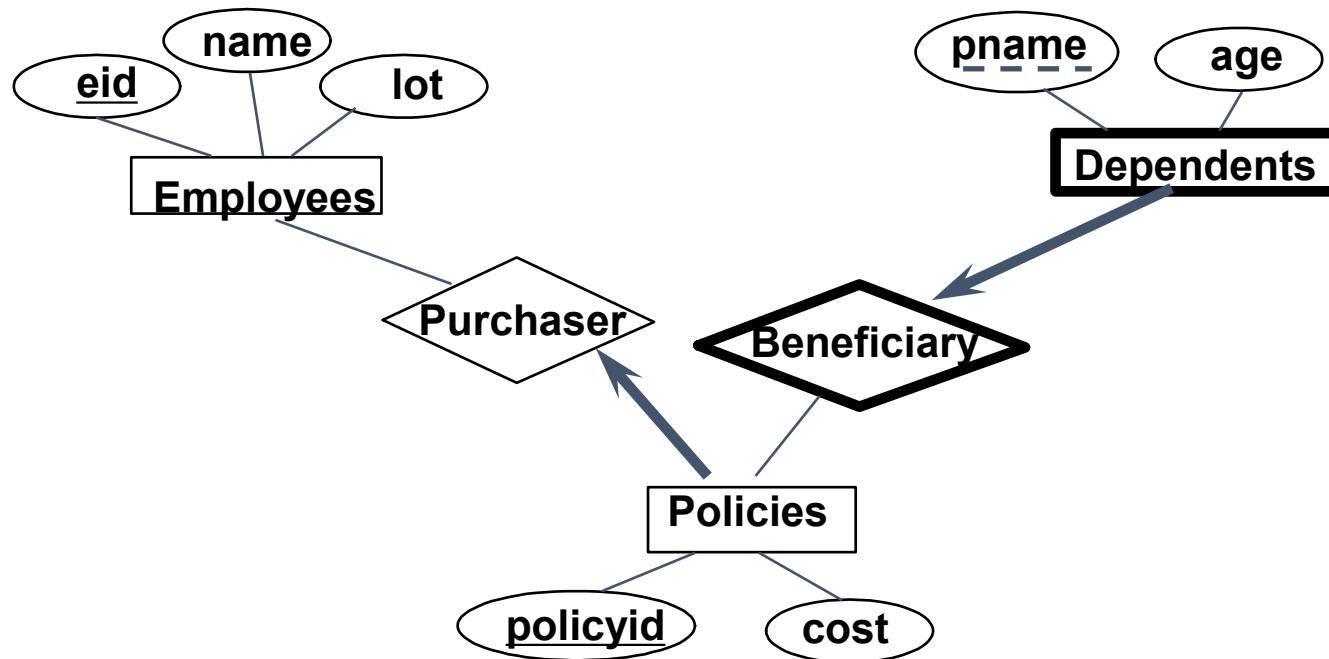


- 追加条件：
 - 一つの保険を複数の従業員が所有することを許さない
 - 保険には必ず所有の従業員がいる
 - 扶養家族は弱実体であり、`pname`と`policyid`で一意に決める

Bad design

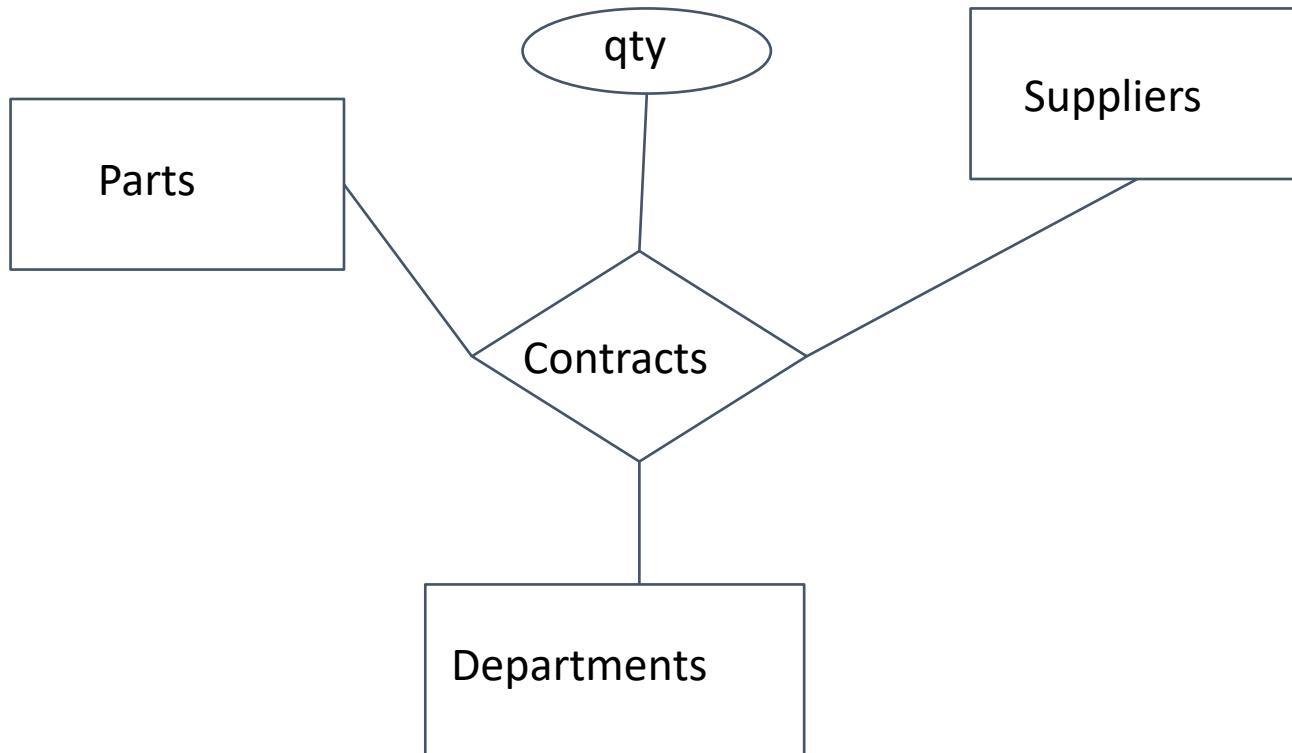
二項関連 vs. 三項関連

Better design



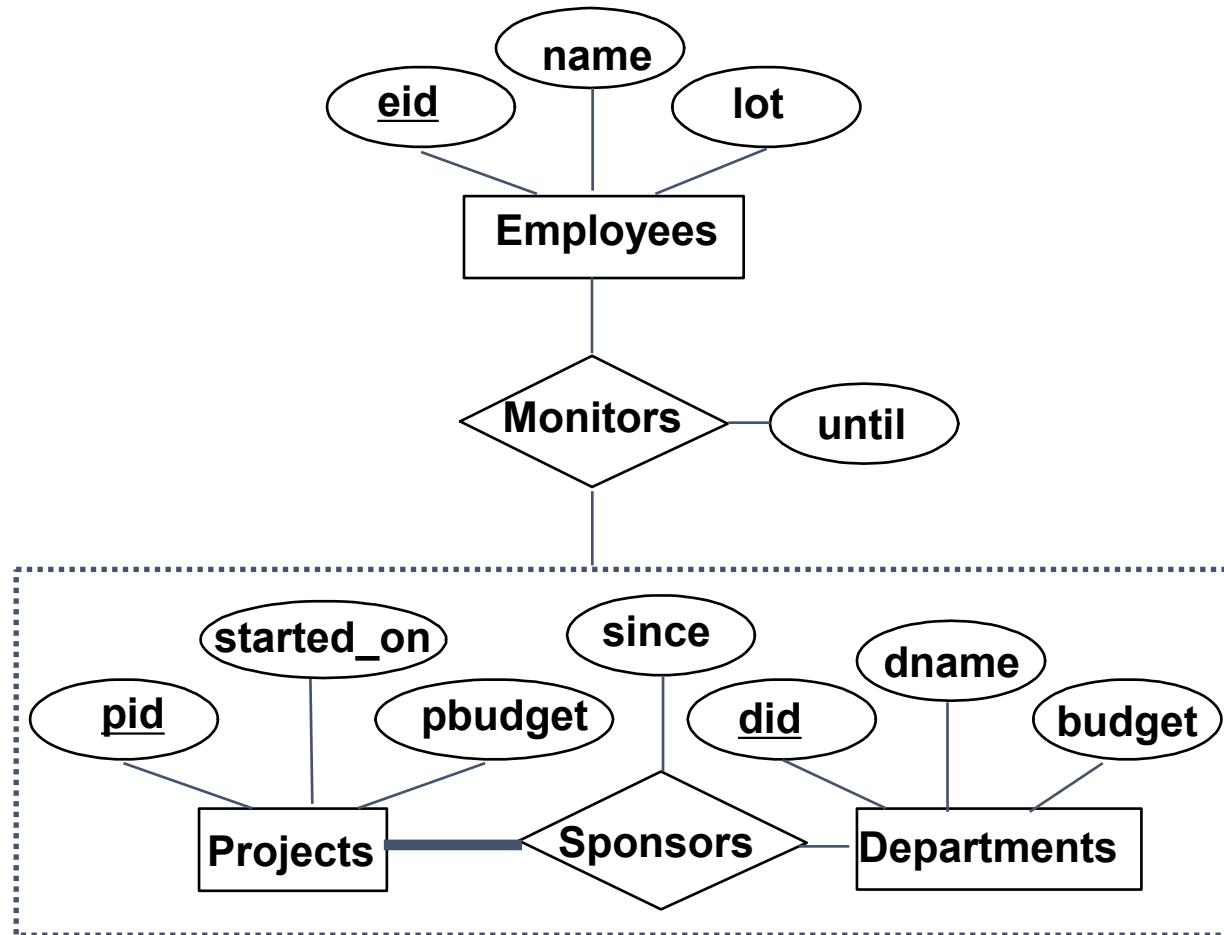
二項関連 VS 三項関連

一つの契約(contract)は、Supplier sが数量qtyのPart pをdepartment dに供給することを示す

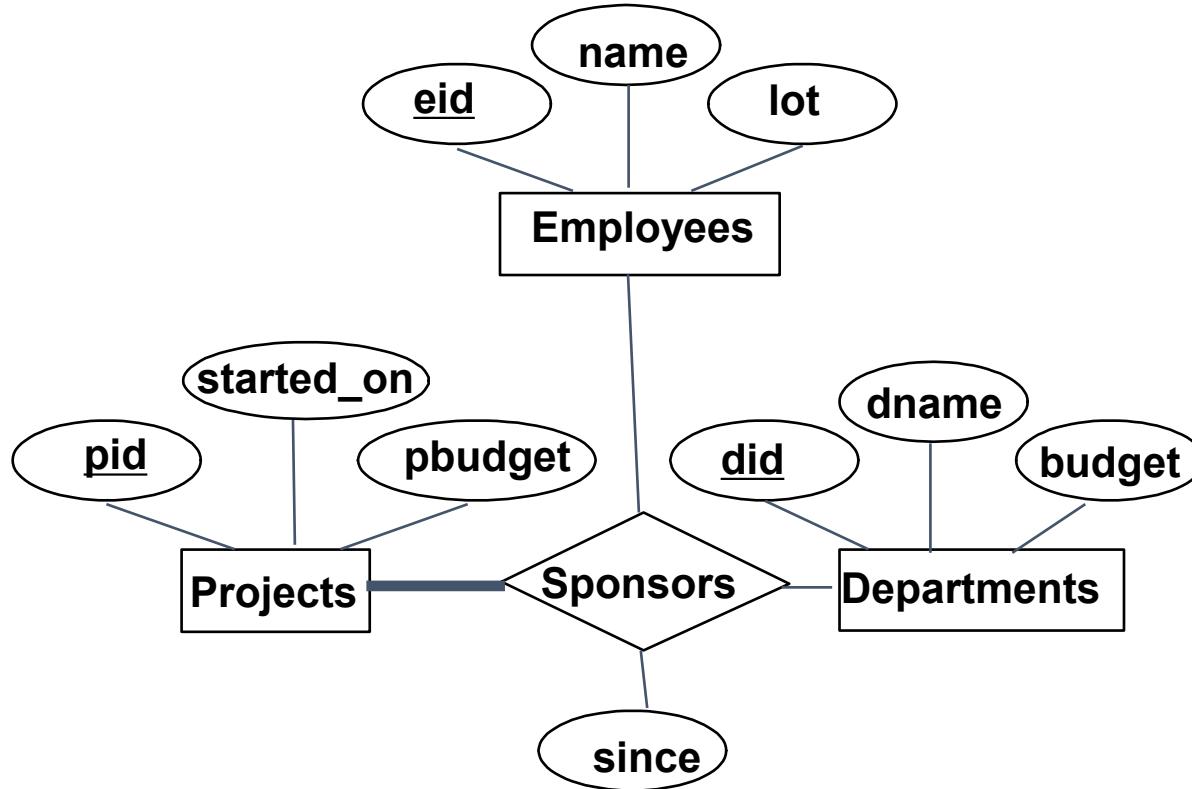


複数の二項関係を使ってこれをうまく表現できるか？

集約 vs. 三項関係



集約 vs. 三項關係 (cont.)



まとめ

概念設計

- 要求分析から開始
- 高レベルの記述

ERモデルは概念設計によく利用される

- 実体、関連、属性
- 弱実体、ISA階層、集約
- 一貫性制約
 - キー制約、参加制約、overlap/covering制約 (ISA階層)
 - 外部キー制約を暗黙に表現可能な場合がある
 - 関数従属性を表現できない
- 主観的な設計

関係モデル

次回