CSS325 Database System
Final Exam Summary

Normalization

- prevent the possible occurance of anomalies
  - → insert anomaly → delete anomaly → update anomaly
- ...→ a technique for producing a set of suitable relations
  - ↳ minimal necessary attribute)
  - ↳ minimal redundancy ; attrb. represented only once.

- 4 common form of normalization (NF : Normal form)
  - ↳ 1NF ↳ 2NF ↳ 3NF ↳ BCNF (Boyce-Codd)

- Redundant data : repeated data for every occurance.
- Properties of decomposition
  - ↳ Lossless-join : find instance of orijinal from smaller (corresponing) relation
  - ↳ Dependency preservation: enforce constraint on orijnal by enforce some constraint on smaller.

- functional Dependency.
  - ♥ what attribute depends on attribute
  - A → B : if each value of A is associated w/ exactly 1 of B
  - → characteristics : - have 1:1 or M:1 relationship b/w attrb.
    - hold for all time
    - non-trivial.

↗ depends on entire PK
- ♥ full functional dependency
- ♥ partial functional dependency
  - ↳ depends on subset of PK
- ♥ transitive dependency
  - ↳ depend on each others.

A	B	C	D	E
a	b	z	w	q
e	b	r	w	p
a	d	z	w	t
e	d	r	w	q
a	f	z	s	t
e	f	r	s	t

transitive depend.

$A \to C$, $C \to A$, $B \to D$, $(A,B) \to E$

partial f^n depend. — full f^n depend.

- how to identify primary key using functional dependency
  - ↳ identify group of attrb. that uniquely identifies each tuple in the tables.
  - ↳ in this case, PK is (A,B)

- normalization step.

UNF
↓ : Ungroup : flatten the table
1NF
↓ : Primary + Partial dependency
2NF
↓ : Transitive Dependency
3NF

(A, B, C, D, E)
(A, B, C, E), (B, D)
(A, B, E), (A, C), (B, D)

BCNF: non-prime attrb. → prime-attrb.
(determinant)

A	B	C	D	E	F	G
a1	b1	c1	d1	e1	f1	g1
a1	b2	c2	d2	e2	f1	g2
a1	b2	c3	d3	e3	f1	g2
a1	b2	c4	d4	e4	f1	g3
a2	b1	c1	d1	e1	f2	g1
a2	b2	c1	d2	e4	f2	g2
a2	b2	c3	d3	e2	f2	g2
a2	b2	c4	d4	e4	f2	g3
a3	b1	c1	d1	e1	f2	g1
a3	b2	c2	d2	e2	f2	g2
a3	b2	c1	d3	e6	f2	g2
a3	b2	c4	d4	e4	f2	g3

functional dependencies

$A \to F$ ✓
$D \to B$ ✓
$D \to G$ ✓
$E \to B$
$G \to B$
$(A, D) \to C, E$ full fk

(partial)

PK: (A, D)

1NF: (A, B, C, D, E, F, G)

2NF: (A, C, D, E), (A, F), (D, B, G)

3NF: (A, C, D, E) (A, F), (D, B, G)

BCNF: (A, C, D, E), (A, F), (D, G), (G, B)

A	f
a1	f1
a2	f2
a3	f2

D	B	G
d1	b1	g1
d2	b2	g2
d3	b2	g2
d4	b2	g3

D	G
d1	g1
d2	g2
d3	g2
d4	g3

G	B
g1	b1
g2	b2
g3	b2

A	C	D	E
a1	c1	d1	e1
a1	c2	d2	e2
a1	c3	d3	e3
a1	c4	d4	e4
a2	c1	d1	e1
a2	c1	d2	e4
a2	c3	d3	e2
a2	c4	d4	e4
a3	c1	d1	e1
a3	c2	d2	e2
a3	c1	d3	e6
a3	c4	d4	e4

• Stored Procedure.

• set of SQL statements that can be stored in the server

Function / Procedure / Triggers

↳ user-defined function
↳ must return one value of given type

↳ return 1 or more dataset as OUT

↳ depend on the event
event: INSERT, DELETE, UPDATE × BEFORE / AFTER

• DECLARATION

→ Function

```
DELIMITER $$
CREATE FUNCTION fn_name(param1 DataType, ...)
RETURNS DataType [NOT] DETERMINISTIC.
BEGIN
    DECLARE result DataType;
    Statements;
    RETURN result;
END$$
DELIMITER ;
```

→ Procedure

```
DELIMITER $$
CREATE PROCEDURE pd_name([IN/OUT] param DataType,...)
BEGIN
    Statements;
END $$
DELIMITER ;
```

• While

```
WHILE condition DO
    statements;
END WHILE;
```

→ Triggers

```
DELIMITER $$
CREATE TRIGGER tg_name
[BEFORE / AFTER] [INSERT / DELETE / UPDATE]
ON table_name FOR EACH ROW
BEGIN
    statements;
END $$
DELIMITER ;
```

↳ new.*: new values
↳ old.*: old values

• Declare variable

```
DECLARE variable_name DataType [DEFAULT initial value];   -- initial value: optional
```

• Flow Controls

• If / else

```
IF condition THEN
    statements;
ELSEIF condition THEN
    statements;
ELSE
    statement;
END IF;
```

• Case

```
CASE variable_name
    WHEN value1 THEN
        statements;
    WHEN value2 THEN
        statements;
    ELSE
        statements;
END CASE;
```

• Cursor

```
DECLARE c-name CURSOR FOR SELECT ....;   -- iterate through result
DECLARE CONTINUE HANDLER FOR NOT FOUND statement;   -- when no result from cursor
OPEN c-name;   -- open cursor for use
FETCH c-name INTO variable_name [attrb.]   -- assign value from cursor to variable
CLOSE c-name;   -- close cursor after used.
```

Relational Algebra & Relational Calculus

- **Definition:** Relation = table w/ rows & columns
  Attributes = column name
  Domain = set of allowable values
  Tuple = a row — Cardinality = # of tuples
  Degree = # of attribute
  Relational Database = a collection of relations

- **Cartesian Product** $A \times B = \{(x,y) \mid x \in A \wedge y \in B\}$
  - subset of cartesian product is a relation

- **Relational Algebra (RA)**

Unary Relational Operation
- σ : SELECT (select some rows based on condition)
- π : PROJECT (select some columns)
- ρ : RENAME (don't worry we aren't going to use it)

Set Theory Operation
- $\cup$: UNION
- $\cap$: INTERSECT
- $-$: DIFFERENCE
- $\times$: CARTESIAN PRODUCT

Join Operation
- $R \bowtie_F S$: Theta-join $= \sigma_F(R \times S)$
  * $F = \{>, \geq, <, \leq, =, \sim=\}$
- $R \bowtie_= S$: Equi-join

JOIN (cont.)
- $R \bowtie S$: Natural Join (a.k.a. inner join on common attribute)
- $R \bowtie S$: LEFT Join on common attribute
- $R \bowtie S$: RIGHT Join on common attribute
- $R \rhd_F S$: Semi Join $= \Pi_A(R \bowtie_F S)$ (basically, join but show attributes in R only)
- $R \div S$: Division (Select R if match every tuple in S on attribute C)

Aggregated function
- $\Im_{\text{max attrib.}}(R)$: MAX value in that attribute (MIN / AVERAGE / SUM / COUNT)
- $A\Im_{\text{max attrib}}(R)$: MAX value in that attribute GROUP BY attribute A

- **Relational Calculus (RC)** $\{x \mid P(x)\}$ (x: proposition; P(x): predicate)

Quantifiers
- $\exists$: there exist (existential quantifier)
- $\forall$: for all (universal quantifier)

- Tuple-oriented RC

$\{S.a, \ldots \mid F(S_1, \ldots S_n)\}$

S → relation

- Domain-oriented RC

$\{d_1, \ldots \mid F(d_1, \ldots)\}$

d → domain

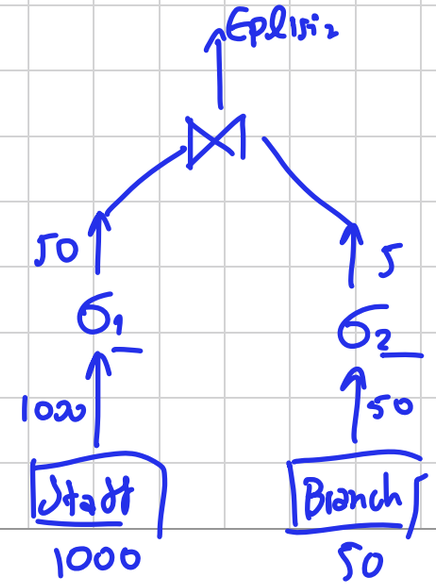
ex List the names of all managers who earn more than £25K

Tuple : $\{S.fName, S.lName \mid Staff(S) \wedge$
$S.position = 'Manager' \wedge$
$S.salary > 25000\}$

Domain : $\{fN, lN \mid (\exists sN, posn, sex, DOB, sal, bN)$
$(Staff(sN, fN, lN, posn, sex, DOB, sal, bN)$
$\wedge posn = 'manager' \wedge sal > 25000\}$

---

(3)



Initial : 1000 + 50
σ_1 : 50
σ_2 : 5
$\bowtie$: 50 + 5
Total : 1,160

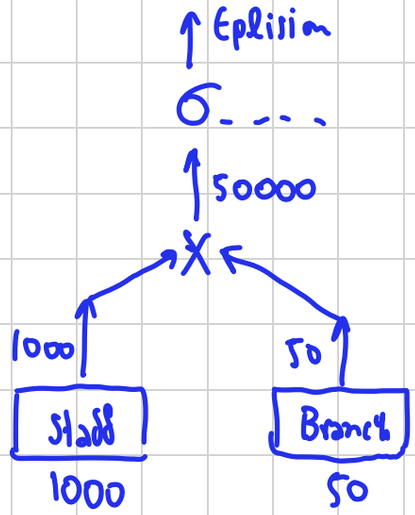
Query Processing

"find the efficient & correct execution strategy of query written in high-level language"

(1) $\sigma_{(position='Manager') \wedge (city='London') \wedge (staff.bno=branch.bno)}$ (Staff × Branch)
(2) $\sigma_{(position='Manager') \wedge (city='London')}$ (Staff $|><|_{staff.bno=branch.bno}$ Branch)
(3) $(\sigma_{position='Manager'}(Staff))$ $|><|_{staff.bno=branch.bno}$ $(\sigma_{city='London'}$ (Branch))

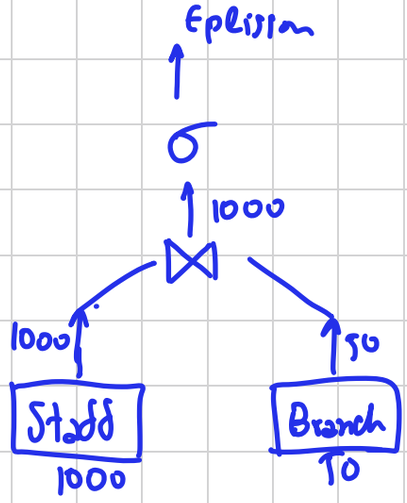
- **1000 tuples in Staff; 50 tuples in Branch;**
- **50 Managers; 5 London branches;**
- **No indexes or sort keys;**
- **Results of any intermediate operations stored on disk;**
- **Cost of the final write is ignored;**
- **Tuples are accessed one at a time.**

(1)



initial : 1000 + 50
after X : 50000 ; (1000 × 50)
σ : 50 000
= 1000 + 50 + (2 × 50,000)
= 101,050

(2)



Initial : 1000 + 50
⋈ : 1000
σ : 1000
= 3,050