Relational Algebra

5 Basic Operations

S1

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

S2

sid	sname	rating	age
28	yuppy	9	35
31	lubber	8	55
44	guppy	5	35
58	rusty	10	35

R1

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

1.Selection σ

Horizontal filtering (tuples)

$\sigma_{rating>8}(S2) = \sigma_{rating>8 \wedge age<50>}(S2) =$

sid	sname	rating	age
28	yuppy	9	35
58	rusty	10	35

Conditions: $<, >, <=, >=, =, !=$

Clauses: and=∧, or=∨

2.Projection π

Vertical filtering (columns); remove duplicates

$\pi_{age}(S2) =$

age
35
55

$\pi_{sname,rating}(\sigma_{rating>9}(S2)) =$

sname	rating
yuppy	9
rusty	10

3.Cross-product $\times$

combine two tables together with all possible combinations

$R1 \times S1 =$

sid	sname	rating	age	sid	bid	day
22	dustin	7	45	22	101	10/10/96
31	lubber	8	55	22	101	10/10/96
58	rusty	10	35	22	101	10/10/96
22	dustin	7	35	58	103	11/12/96
31	lubber	8	55	58	103	11/12/96
58	rusty	10	35	58	103	11/12/96

Deal with conflicting names: rename the attributes

$\rho(C(1->sid1, 5->sid2), R1 \times S1) =$

sid1	sname	rating	age	sid2	bid	day
22	dustin	7	45	22	101	10/10/96
31	lubber	8	55	22	101	10/10/96
58	rusty	10	35	22	101	10/10/96
22	dustin	7	35	58	103	11/12/96
31	lubber	8	55	58	103	11/12/96
58	rusty	10	35	58	103	11/12/96

4.Set-difference —

filtering tuples; Require Union-Compatible (1.same #fields; 2.same types); Non-symmetrical

$S1 - S2 =$

sid	sname	rating	age
22	dustin	7	45

$S2 - S1 =$

sid	sname	rating	age
28	yuppy	9	35
44	guppy	5	35

5.Union ∪

combine tuples; Require Union-Compatible (1.same #fields; 2.same types); remove duplicates

$S1 \cup S2 =$

sid	sname	rating	age
22	dustin	7	45
28	yuppy	9	35
31	lubber	8	55
44	guppy	5	35
58	rusty	10	35

Compound Operations

1.Intersection ∩

$A \cap B = A - (A - B)$

$S1 \cap S2 = S2 \cap S1 =$

sid	sname	rating	age
31	lubber	8	55
58	rusty	10	35

2.Join ⋈

(1) natural join

1. Compute $R \times S$;
2. equal values attributes;
3. project unique attributes

$S1 \bowtie R1 = \pi_{...}(\sigma_{unique?}(S1 \times R1)) =$

sid	sname	rating	age	bid	day
22	dustin	7	45	101	10/10/96
58	rusty	10	35	103	11/12/96

(2) condition join (/ theta-join)

$R \bowtie_c S = \sigma_c(R \times S)$

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

sid	sname	rating	age	sid	bid	day
22	dustin	7	35	58	103	11/12/96
31	lubber	8	55	58	103	11/12/96

Equi-Join: condition c contains only equalities (e.g. S1.sid = R1.sid)

Still different from natural join, since we can choose the attribute.

> e.g. (1) Find names of sailors who have reserved boat #103
>
> solution 1: $\pi_{sname}((\sigma_{bid=103} Reserves) \bowtie Sailors)$
>
> solution 2: $\pi_{sname}(\sigma_{bid=103}(Reserves \bowtie Sailors))$

> e.g. (2) Find all pairs of sailors in which the older sailor has a lower rating
>
> solution 1:
> $\pi_{sname}((\sigma_{(S1.age<S2.age \wedge S1.rating>S2.rating) \vee (S1.age>S2.age \wedge S1.rating<S2.rating)} S1) \bowtie S2)$
>
> solution 2:
> $\pi_{sname}(\sigma_{(S1.age<S2.age \wedge S1.rating>S2.rating) \vee (S1.age>S2.age \wedge S1.rating<S2.rating)}(S1 \bowtie S2))$

SQL

SQL / SEQUEL

DBMS and SQL support **CRUD**: Create, Read, Update, Delete commands

DDL (Definition): `CREATE, ALTER, DROP, TRUNCATE, RENAME`

DML (Manipulation): Comparison & Logic Operators, Set Operations, Subquery, Multiple record INSERTs, INSERT from a table, `VIEW, SELECT, INSERT, DELETE, UPDATE, REPLACE`

DCL (Control): Users and permissions, e.g. `CREAT USER, DROP USER, GRANT, REVOKE, SET PASSWORD`

Database Administration Statements:

Database Administration: e.g. `BACKUP TABLE, RESTORE TABLE, ANALYZE TABLE`

Miscellaneous: e.g. `DESCRIBE tablename, USE db_name`

others: Transaction Control...

Case sensitivity

- SQL keywords are case insensitive
- Table names are Operation System Sensitive
- Field names are case insensitive
  SQL is able to do math expressions

Data Manipulation Language (DML)

Comparison & Logic Operators & Set Operations

1. **Comparison**: =, <, >, <=, >=, <> OR !=
2. **Logic**: `NOT > AND > OR` (use '()' would be better)
3. **Set Operation**: `UNION` (either table, expressions is fine), `UNION ALL` (exists duplication; SQL), `INTERSECT`

```
SELECT  Employee.Name, EmployeeType
        FROM Employee INNER JOIN Hourly
        ON  Employee.ID = Hourly.ID
UNION
SELECT  Employee.Name, EmployeeType
        FROM Employee INNER JOIN Salaried
        ON  Employee.ID = Salaried.ID;
```

Query Nesting

nesting subqueries, a nested query is simply another select query;

Retionale: All select queries return a table set of data.

1. `IN/NOT IN` (if changeable, joins are faster than ins)

- List the BuyerID, Name and Phone number for all bidders on artefact 1

```
SELECT * FROM Buyer
WHERE BuyerID IN (SELECT BuyerID FROM Offer
                  WHERE ArtefactID = 1)
```

Equals to

```
SELECT BuyerID, Name and Phone
FROM Buyer NATURAL JOIN Offer
WHERE ArtefactID = 1
```

join faster than query!

This is a more efficient way

2. `ANY`, `ALL`

- **All**: must satisfy **all** inner conditions

```
SELECT empno, sal
FROM emp
WHERE sal > ALL (200, 300, 400);
```

ALL用在最后where里

Equiv.

```
SELECT empno, sal
FROM emp
WHERE sal > 200 AND sal > 300 AND sal> 400;
```

- **Any**: must satisfy **at least one of** the inner conditions (any of)

```
SELECT empno, sal
FROM emp
WHERE sal > ANY (200, 300, 400);
```

Equiv.

```
SELECT empno, sal
FROM emp
WHERE sal > 200 OR sal > 300 OR sal> 400;
```

- **Exists**: the inner query returns **at least one** record

```
SELECT  empid, first_name, last_name
FROM employees AS E
WHERE
   EXISTS( SELECT  * FROM dependents AS D WHERE D. empid = E. empid);
```

"Print all employees who have at least one dependent"

3. `EXISTS`: ①对外层的表的每一个tuple ②检查里层的表是否有满足条件的tuple ③if not exists: move on to next tuple ④if exists: SELECT 操作这个tuple

- Returns true if the subquery returns one or more records
- **Example**: *List the BuyerID, Name and Phone number for all bidders on artefact 1*

```
SELECT * FROM Buyer  WHERE EXISTS
(SELECT * FROM Offer  WHERE Buyer.BuyerID = Offer.BuyerID
                      AND ArtefactID = 1)
```

Offer

SellerID	ArtefactID	BuyerID	Date	Amount	Acceptance
1	1	1	2012-06-20	81223.23	N
1	1	2	2012-06-20	82223.23	N
2	2	1	2012-06-20	19.95	N
2	2	2	2012-06-20	23.00	N

Buyer

BuyerID	Name	Phone
1	Maggie	0333333333
2	Nicole	0444444444
3	Oleg	0555555555

对外层的(Buyer)的每一个tuple:
检查在里层的表(Offer) 是否有满足条件的tuple
if not exists: move on to next tuple
if exists: 再从外层的(Buyer)的那个tuple进行操作

Result

BuyerID	Name	Phone
1	Maggie	0333333333
2	Nicole	0444444444

INSERT, UPDATE, REPLACE, DELETE

1. `INSERT`:

```
# table must already exist
INSERT INTO NewEmployee
SELECT * FROM Employee;

# Without specifying fields: All columns must be specified
INSERT INTO Employee VALUES
```

```
    (DEFAULT, "A", "A's Addr", "2012-02-02", NULL, "S"),
    (DEFAULT, "B", "B's Addr", "2012-02-02", NULL, "S"),
    (DEFAULT, "C", "C's Addr", "2012-02-02", NULL, "S")
;

# Specifying fields:
INSERT INTO Employee
    (Name, Address, DateHired, EmployeeType)
    VALUES
    (DEFAULT, "A", "A's Addr", "2012-02-02", "S"),
    (DEFAULT, "B", "B's Addr", "2012-02-02", "S"),
    (DEFAULT, "C", "C's Addr", "2012-02-02", "S")
;
```

2. `UPDATE`: Specifying a `WHERE` clause is important; Order of statements is important

```
# update the whole table
UPDATE Hourly
    SET HourlyRate = HourlyRate * 1.10;

# wrong order
UPDATE Salaried
    SET AnnualSalary = AnnualSalary * 1.05
    WHERE AnnualSalary <= 10000;
UPDATE Salaried
    SET AnnualSalary = AnnualSalary * 1.05
    WHERE AnnualSalary > 10000;

# Should use CASE
UPDATE Salaried
    SET AnnualSalary =
        CASE
            WHEN AnnualSalary <= 10000
            THEN AnnualSalary * 1.05
            ELSE AnnualSalary * 1.10
        END;
```

3. `REPLACE`: used when you are unsure whether a record exists or not; if not exists, like INSERT. O.W. like UPDATE
4. `DELETE`: for FKs:

   ON DELETE CASCADE: 删掉所有指向它的records

   ON DELETE RESTRICT: 禁止我们删除该条，如果存在指向它的records

```
# delete all records
DELETE FROM Employee;

# delete specified records
SELETE FROM Employee
    WHERE Name = "Grace";
```

Views

Relation that is not in the physical models, but is made available to the user as a virtual relation.

Once a view is defined, the definition is stored in the DB (not data, but metadata-schema information); So can be used as other tables

Every time executed is running the complex definition

Pros

1. Hide the query comlexity from users
2. Hide data from users (different users are set to use different views)

```
CREATE VIEW EmpPay AS
SELECT  Employee.ID, Employee.Name, DateHired,
        EmployeeType, HourlyRate AS Pay
        FROM Employee INNER JOIN Hourly
        ON  Employee.ID = Hourly.ID
UNION
SELECT  Employee.ID, Employee.Name, DateHired,
        EmployeeType, AnnualSalary  AS Pay
        FROM Employee INNER JOIN Salaried
        ON  Employee.ID = Salaried.ID
UNION
SELECT  Employee.ID, Employee.Name, DateHired,
        EmployeeType, BillingRate  AS Pay
        FROM Employee INNER JOIN Consultant
        ON    Employee.ID = Consultant.ID;
```

DDL Commands

1. `CREATE` Statement (with FK)

```
CREATE TABLE Account (
  AccountID             smallint        auto_increment,
  AccountName           varchar(100)    NOT NULL,
  OutstandingBalance    DECIMAL(10,2)   NOT NULL,
  CustomerID            smallint        NOT NULL,
  PRIMARY KEY  (AccountID),
  FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)
        ON DELETE RESTRICT
        ON UPDATE CASCADE
);
```

2. `ALTER`: add or remove attributes (columns) from a relation

```
ALTER TABLE TableName ADD AttributeName AttributeType
ALTER TABLE TableName DROP AttributeName
```

3. `RENAME`: rename the table

```
RENAME TABLE CurrentTableName TO NewTableName
```

4. `TRUNCATE`: like DELETE, but faster and can't ROLL BACK (recover from backup)
5. `DROP`: removes the data and the relation, can't UNDO (recover from backup)

```
DROP TABLE TableName
```

SQL Usage

1. USE the database design as a MAP to help you when you are formulating queries
2. USE the structure of the SELECT statement as a template