CSE 412 Database Management

Dr. Bharatesh Chakravarthi (BC)

Assistant Teaching Professor

School of Computing and Augmented Intelligence

Arizona State University

Review of SQL

Review of Relational Algebra Expression and SQL

- ***Selection (σ)*** *– Filters rows based on conditions.*
- ***Projection (π)*** *– Selects specific columns.*
- ***Cartesian Product (×)*** *– Combines all rows from two tables.*
- ***Join (⋈)*** *– Combines related rows from two tables based on a condition.*
- ***Natural Join (⋈)*** *– Automatically joins on common attributes.*
- ***Union (∪)*** *– Combines two sets of tuples.*
- ***Intersection (∩)*** *– Returns common tuples in both sets.*
- ***Difference (-)*** *– Returns tuples from one set that are not in another.*
- ***Renaming (ρ)*** *– Changes the attribute names of a relation.*

Review of Relational Algebra Expression and SQL

Simple Selection (σ)

$$\sigma_{age>10}(User)$$

SELECT * FROM "user" WHERE age > 10;

User

uid	name	age	pop
142	Bart	10	0.9
123	Milhouse	10	0.2
857	Lisa	8	0.7
456	Ralph	8	0.3
…	…	…	…

Group

gid	name
abc	Book Club
gov	Student Government
dps	Dead Putting Society
…	…

Member

uid	gid
142	dps
123	gov
857	abc
857	gov
456	abc
456	gov
…	…

Review of Relational Algebra Expression and SQL

Projection (π)

$$\pi_{name,\ age}(User)$$

SELECT name, age FROM "user";

User

uid	name	age	pop
142	Bart	10	0.9
123	Milhouse	10	0.2
857	Lisa	8	0.7
456	Ralph	8	0.3
…	…	…	…

Group

gid	name
abc	Book Club
gov	Student Government
dps	Dead Putting Society
…	…

Member

uid	gid
142	dps
123	gov
857	abc
857	gov
456	abc
456	gov
…	…

Review of Relational Algebra Expression and SQL

Join (⋈)

$$\pi_{\text{user.name, group.name}}((\text{User} \bowtie_{\text{uid=mid}} \text{Member}) \bowtie_{\text{gid=gid}} \text{Group})$$

User

uid	name	age	pop
142	Bart	10	0.9
123	Milhouse	10	0.2
857	Lisa	8	0.7
456	Ralph	8	0.3
…	…	…	…

```
SELECT "user".name, "group".name
FROM "user“
JOIN "member" ON "user".uid = "member".mid
JOIN "group" ON "member".gid = "group".gid;
```

Group

gid	name
abc	Book Club
gov	Student Government
dps	Dead Putting Society
…	…

Member

uid	gid
142	dps
123	gov
857	abc
857	gov
456	abc
456	gov
…	…

`User ⋈ (uid = mid) Member` : Joins `User` and `Member` on `uid = mid` .

`(...) ⋈ (gid = gid) Group` : Joins the previous result with `Group` on `gid` .

`πuser.name, group.name (...)` : Selects only `user.name` and `group.name` .

Review of Relational Algebra Expression and SQL

Join (⋈)

$$\pi_{\text{user.name, group.name}}((\text{User} \bowtie_{\text{uid=mid}} \text{Member}) \bowtie_{\text{gid=gid}} \text{Group})$$

User

uid	name	age	pop
142	Bart	10	0.9
123	Milhouse	10	0.2
857	Lisa	8	0.7
456	Ralph	8	0.3
…	…	…	…

```
SELECT "user".name, "group".name
FROM "user“
JOIN "member" ON "user".uid = "member".mid
JOIN "group" ON "member".gid = "group".gid;
```

Group

gid	name
abc	Book Club
gov	Student Government
dps	Dead Putting Society
…	…

Member

uid	gid
142	dps
123	gov
857	abc
857	gov
456	abc
456	gov
…	…

`User ⋈ (uid = mid) Member` : Joins `User` and `Member` on `uid = mid` .

`(...) ⋈ (gid = gid) Group` : Joins the previous result with `Group` on `gid` .

`πuser.name, group.name (...)` : Selects only `user.name` and `group.name` .

Review of Relational Algebra Expression and SQL

Set Operations (Union)

$$\pi_{uid}(User) \cup \pi_{mid}(Member)$$

User

uid	name	age	pop
142	Bart	10	0.9
123	Milhouse	10	0.2
857	Lisa	8	0.7
456	Ralph	8	0.3
…	…	…	…

SELECT uid FROM "user “

UNION

SELECT mid FROM "member";

Group

gid	name
abc	Book Club
gov	Student Government
dps	Dead Putting Society
…	…

Member

uid	gid
142	dps
123	gov
857	abc
857	gov
456	abc
456	gov
…	…

`πuid(User)` : Selects the `uid` column from `User`.

`πmid(Member)` : Selects the `mid` column from `Member`.

`∪` : Performs the **union** operation.

Review of Relational Algebra Expression and SQL

Set Operations (Difference)

$$\pi_{uid}(User) - \pi_{mid}(Member)$$

User

uid	name	age	pop
142	Bart	10	0.9
123	Milhouse	10	0.2
857	Lisa	8	0.7
456	Ralph	8	0.3
…	…	…	…

SELECT uid FROM "user “

EXCEPT

SELECT mid FROM "member";

Group

gid	name
abc	Book Club
gov	Student Government
dps	Dead Putting Society
…	…

Member

uid	gid
142	dps
123	gov
857	abc
857	gov
456	abc
456	gov
…	…

`πuid(User)` : Selects `uid` from `User` .

`πmid(Member)` : Selects `mid` from `Member` .

`-` : Performs the **difference** operation, returning users who are not in `Member` .

Review of Relational Algebra Expression and SQL

Set Operations (Intersection)

$$\pi_{uid}(User) \cap \pi_{mid}(Member)$$

User

uid	name	age	pop
142	Bart	10	0.9
123	Milhouse	10	0.2
857	Lisa	8	0.7
456	Ralph	8	0.3
…	…	…	…

SELECT uid FROM "user “

INTERSECT

SELECT mid FROM "member";

Group

gid	name
abc	Book Club
gov	Student Government
dps	Dead Putting Society
…	…

Member

uid	gid
142	dps
123	gov
857	abc
857	gov
456	abc
456	gov
…	…

`πuid(User)` → Selects only the `uid` column from the **User** table.

`πmid(Member)` → Selects only the `mid` column from the **Member** table.

`∩` **(Intersection)** → Returns only the common values present in both results, i.e., user IDs that exist in both tables.

Review of Relational Algebra Expression and SQL

Order By

Unlike SQL, relational algebra does not have a built-in ORDER BY operator becau traditional relational algebra focuses on unordered sets.

However, in extended relational algebra, sorting can be represented using the ***τ (tau)*** *operator.*

The general form is:

$$\tau_{A\ 1\ (order1),\ A\ 2\ (order2),\ \ldots}(R)$$

$A_1, A_2, \ldots$ → Attributes used for sorting.

`order1, order2, ...` → Sorting order (**ASC** for ascending, **DESC** for descending).

R → The relation (table) being sorted.

User

uid	name	age	pop
142	Bart	10	0.9
123	Milhouse	10	0.2
857	Lisa	8	0.7
456	Ralph	8	0.3
…	…	…	…

Group

gid	name
abc	Book Club
gov	Student Government
dps	Dead Putting Society
…	…

Member

uid	gid
142	dps
123	gov
857	abc
857	gov
456	abc
456	gov
…	…

Review of Relational Algebra Expression and SQL

Order By

$$\tau_{\textit{pop DESC, name ASC}}(\textit{User})$$

User

uid	name	age	pop
142	Bart	10	0.9
123	Milhouse	10	0.2
857	Lisa	8	0.7
456	Ralph	8	0.3
…	…	…	…

SELECT uid, name, age, pop

FROM "user“

ORDER BY pop DESC, name ASC;

Group

gid	name
abc	Book Club
gov	Student Government
dps	Dead Putting Society
…	…

Member

uid	gid
142	dps
123	gov
857	abc
857	gov
456	abc
456	gov
…	…

`τ` **(tau) is the sorting operator** in extended relational algebra.

`pop DESC` → Orders results by `popularity` in descending order.

`name ASC` → If two users have the same popularity, it sorts them by `name` in ascending order.

`User` → The relation (table) being sorted.

Review of Relational Algebra Expression and SQL

Group By

User

uid	name	age	pop
142	Bart	10	0.9
123	Milhouse	10	0.2
857	Lisa	8	0.7
456	Ralph	8	0.3
…	…	…	…

In relational algebra, γ (gamma) represents the grouping and aggregation operator, similar to GROUP BY in SQL.

The general form is:

$$\gamma_{A,F(B)}(R)$$

Group

gid	name
abc	Book Club
gov	Student Government
dps	Dead Putting Society
…	…

Member

uid	gid
142	dps
123	gov
857	abc
857	gov
456	abc
456	gov
…	…

A → Attribute(s) to **group by** (like `GROUP BY` in SQL).

$F(B)$ → An **aggregate function** applied to attribute B (like `COUNT`, `SUM`, `AVG`, etc.).

R → The **relation (table)** on which the operation is performed.

Review of Relational Algebra Expression and SQL

Group By

$$\gamma_{gid,COUNT(mid)}(Member)$$

User

uid	name	age	pop
142	Bart	10	0.9
123	Milhouse	10	0.2
857	Lisa	8	0.7
456	Ralph	8	0.3
…	…	…	…

Group

gid	name
abc	Book Club
gov	Student Government
dps	Dead Putting Society
…	…

```
SELECT gid, COUNT(mid) AS member_count
FROM "member“
GROUP BY gid;
```

Member

uid	gid
142	dps
123	gov
857	abc
857	gov
456	abc
456	gov
…	…

`γ` **(Grouping Operator)** → Used for grouping in relational algebra.

`gid` → The column by which we group the records (i.e., group ID).

`COUNT(mid)` → Counts the number of members (`mid`) in each group.

Review of Relational Algebra Expression and SQL

User

uid	name	age	pop
142	Bart	10	0.9
123	Milhouse	10	0.2
857	Lisa	8	0.7
456	Ralph	8	0.3
…	…	…	…

Having

The HAVING clause is used to filter the results of a GROUP BY query based on an aggregate function. It works similarly to the WHERE clause but applies to the results after grouping.

Group

gid	name
abc	Book Club
gov	Student Government
dps	Dead Putting Society
…	…

Member

uid	gid
142	dps
123	gov
857	abc
857	gov
456	abc
456	gov
…	…

Review of Relational Algebra Expression and SQL

Having

$$\sigma_{COUNT(mid)>1}(\gamma_{gid,COUNT(mid)}(Member))$$

User

uid	name	age	pop
142	Bart	10	0.9
123	Milhouse	10	0.2
857	Lisa	8	0.7
456	Ralph	8	0.3
…	…	…	…

Group

gid	name
abc	Book Club
gov	Student Government
dps	Dead Putting Society
…	…

```
SELECT gid, COUNT(mid) AS member_count
FROM "member“
GROUP BY gid
HAVING COUNT(mid) > 1;
```

Member

uid	gid
142	dps
123	gov
857	abc
857	gov
456	abc
456	gov
…	…

`γ (gamma)` → Groups by `gid` and calculates the count of `mid` (number of members per group).

`σ (selection)` → Filters the result of the grouping to include only those groups where the count of members is greater than 1.

`Member` → The relation on which the aggregation and filtering are applied.

Review of Relational Algebra Expression and SQL

- ***ORDER BY:*** *Sorts the results of a query based on one or more columns in either ascending (ASC) or descending (DESC) order.*
- ***GROUP BY:*** *Groups rows that have the same values in specified columns into aggregated data, allowing aggregate functions like COUNT, SUM, AVG, etc., to be applied.*
- ***HAVING:*** *Filters the results of a GROUP BY query based on conditions applied to aggregate functions, similar to WHERE but after grouping.*

Review of Relational Algebra Expression and SQL

Another Example …

$$Result = User \bowtie_{uid=mid} Member \bowtie_{gid=gid} Group$$

User

uid	name	age	pop
142	Bart	10	0.9
123	Milhouse	10	0.2
857	Lisa	8	0.7
456	Ralph	8	0.3
…	…	…	…

Group

gid	name
abc	Book Club
gov	Student Government
dps	Dead Putting Society
…	…

Member

uid	gid
142	dps
123	gov
857	abc
857	gov
456	abc
456	gov
…	…

```
SELECT "user".name AS user_name, "group".group_name
FROM "user "
JOIN "member" ON "user".uid = "member".mid
JOIN "group" ON "member".gid = "group".gid;
```

First JOIN: `"user" JOIN "member"` : We join the `User` table and the `Member` table on the `uid` (from the `user` table) and `mid` (from the `member` table). This gives us a list of users and their corresponding `mid` values.

Second JOIN: `"member" JOIN "group"` : We join the resulting table from the previous join with the `Group` table on the `gid` column to retrieve the group names.

Review of Relational Algebra Expression and SQL

- ***IN:*** *Checks if a value matches any value in a list or subquery, returning true if it does.*
- ***EXISTS:*** *Tests whether a subquery returns any rows; returns true if the subquery yields results, and false otherwise.*
- ***AS:*** *Assigns an alias (temporary name) to a table or column in a query for readability or convenience.*
- ***NOT IN:*** *Tests if a value is not present in a specified list or subquery; returns true if the value is not found.*
- ***ALL:*** *Compares a value to all values in a result set or subquery, usually used with operators like =, >, <, etc., to check if the condition holds true for all elements.*
- ***ANY:*** *Compares a value to any value in a result set or subquery, and returns true if the condition holds true for at least one element.*