Lecture 4: SQL (Basic)

CS348 Spring 2025:
Introduction to Database Management

Instructor: **Xiao Hu**

Sections: 001, 002, 003

Announcement

- Assignment 1:
  - Crowdmark and Marmoset are already open
  - Coverage: Lectures 1 – 6 and partially Lecture 7
    - Overview
    - Relational data and relational algebra
    - SQL basic

- In Assignment 1, focus on the correctness (not efficiency) when writing a RA or SQL query

SQL (Structured Query Language)

- Pronounced “S-Q-L” or “sequel”
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987
- A brief history
  - IBM System R (early 1970s)
  - ANSI SQL86
  - ANSI SQL89
  - ANSI SQL92 (SQL2)
  - ANSI SQL99 (SQL3)
  - ANSI SQL 2003 (added OLAP, XML, etc.)
  - ANSI SQL 2006 (added more XML)
  - ANSI SQL 2008,
  - …

SQL is a standard - BUT

- The standard query language supported by most DBMS

- Although SQL is an ANSI/ISO standard, there are different versions of the SQL language
  - Support at least the major commands (such as SELECT, UPDATE, DELETE, INSERT, WHERE) in a similar manner
  - Most DBMS also have their own proprietary extensions or restrictions in addition to the SQL standard!

What can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can enforce constraints in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views
- ……

SQL

- Basic topics:
  - Data-definition language (DDL): define/modify schemas, drop relations
  - Data-manipulation language (DML): query data, and insert/delete/modify tuples
  - Integrity constraints: specify constraints that the data stored in the database must satisfy

- Advanced topics:
  - E.g., triggers, views, indexes, programming, recursion

SQL

- Basic topics:
  - Data-definition language (DDL): define/modify schemas, drop relations

DDL

User (*uid* int, *name* string, *age* int, *pop* float)
Group (*gid* string, *name* string)
Member (*uid* int, *gid* string)

- CREATE TABLE *table_name*
  (..., *column_name column_type*, ...);

```
CREATE TABLE User(uid INT, name VARCHAR(30), age INT, pop
DECIMAL(3,2));
CREATE TABLE Group (gid CHAR(10), name VARCHAR(100));
CREATE TABLE Member (uid INT, gid CHAR(10));
```

- DROP TABLE *table_name*;

```
DROP TABLE User;
DROP TABLE Group;
DROP TABLE Member;
```

Drastic action: deletes ALL info about the table, not just the contents

```
-- everything from -- to the end of line is ignored.
-- SQL is insensitive to white space.
-- SQL is insensitive to case (e.g., ...CREATE... is equivalent to ...create...).
-- semicolon is used at the end of each SQL statement.
```

Post-Lecture

User (*uid* int, *name* string, *age* int, *pop* float)
Group (*gid* string, *name* string)
Member (*uid* int, *gid* string)

- Add or modify attributes

```
ALTER TABLE Member ADD date
```

```
ALTER TABLE Member RENAME date TO mdate
```

```
ALTER TABLE Member DROP mdate
```

SQL

- Basics
  - Data-definition language (DDL): define/modify schemas, drop relations
  - Data-manipulation language (DML): query data
    - SELECT-FROM-WHERE

Basic queries for DML: SFW statement

- SELECT $A_1, A_2, \ldots, A_n$
  FROM $R_1, R_2, \ldots, R_m$
  WHERE $condition$;

- Also called an SPJ (select-project-join) query

- Corresponds to (but not really equivalent to) relational algebra query:

$$\pi_{A_1,A_2,\ldots,A_n}\left(\sigma_{condition}(R_1 \times R_2 \times \cdots \times R_m)\right)$$

Examples

User (*uid* int, *name* string, *age* int, *pop* float)
Group (*gid* string, *name* string)
Member (*uid* int, *gid* string)

- List all rows in the User table

```
SELECT * FROM User;
```

  - * is a short hand for "all columns"

- List name of users under 18 (selection, projection)

```
SELECT name FROM User WHERE age <18;
```

- When was Lisa born?

```
SELECT 2025-age FROM User WHERE name = 'Lisa';
```

  - SELECT list can contain expressions
  - String literals (case sensitive) are enclosed in single quotes

More examples

User (*uid* int, *name* string, *age* int, *pop* float)
Group (*gid* string, *name* string)
Member (*uid* int, *gid* string)

```
SELECT name FROM User WHERE age <18 OR age > 30;
```

```
SELECT name FROM User WHERE pop > 0.9 AND age < 18;
```

```
SELECT age FROM User WHERE NOT pop < 0.9;
```

```
SELECT age FROM User WHERE name IN ('Lisa', 'Bart', 'Alice');
```

```
SELECT age FROM User WHERE name LIKE '%Lisa%';
```

- WHERE clause can use (not limited to the following):
  - logical connectives (AND, OR, NOT)
  - IN specify multiple values
  - LIKE matches a string against a pattern
    - % matches any sequence characters

Example: join

User (*uid* int, *name* string, *age* int, *pop* float)
Group (*gid* string, *name* string)
Member (*uid* int, *gid* string)

- List IDs and names of groups with a user whose name contains "Lisa"

```
SELECT Group.gid, Group.name
        FROM User, Member, Group
        WHERE User.uid = Member.uid
                AND Member.gid = Group.gid
                AND … .;
```

Example: join

User (*uid* int, *name* string, *age* int, *pop* float)
Group (*gid* string, *name* string)
Member (*uid* int, *gid* string)

- List IDs and names of groups with a user whose name contains "Lisa"

```
SELECT Group.gid, Group.name
        FROM User, Member, Group
        WHERE User.uid = Member.uid
                AND Member.gid = Group.gid
                AND User.name LIKE '%Lisa%';
```

  - Okay to omit *table_name* in *table_name.column_name* if *column_name* is unique

Example: rename

User (*uid* int, *name* string, *age* int, *pop* float)
Group (*gid* string, *name* string)
Member (*uid* int, *gid* string)

- IDs of all pairs of users that join some common group
  - Relational algebra query:

$$\pi_{m_1.uid,m_2.uid}\left(\rho_{m_1} Member \bowtie_{\substack{m_1.gid=m_2.gid \\ \wedge\, m_1.uid<m_2.uid}} \rho_{m_2} Member\right)$$

m1.uid≠ m2.uid?

  - SQL (not exactly due to duplicates):

```
SELECT m1.uid AS uid1, m2.uid AS uid2
        FROM Member AS m1, Member AS m2
        WHERE m1.gid = m2.gid AND m1.uid < m2.uid;
```

  - AS keyword is completely optional

two users join two common groups?

A more complicated example

- Names of all groups that Lisa and Alice are both in

Tip: Write the FROM clause first, then WHERE, and then SELECT

User (*uid* int, *name* string, *age* int, *pop* float)
Group (*gid* string, *name* string)
Member (*uid* int, *gid* string)

A more complicated example

- Names of all groups that Lisa and Alice are both in

```
SELECT
        FROM User u1, … , Member m1, …
        WHERE u1.name = 'Lisa' AND …
                AND u1.uid = m1.uid AND …
                AND …;
```

User (*uid* int, *name* string, *age* int, *pop* float)
Group (*gid* string, *name* string)
Member (*uid* int, *gid* string)

A more complicated example

- Names of all groups that Lisa and Alice are both in

```
SELECT
        FROM User u1, User u2, Member m1, Member m2, …
        WHERE u1.name = 'Lisa' AND u2.name = 'Alice'
                AND u1.uid = m1.uid AND u2.uid=m2.uid
                AND …;
```

User (*uid* int, *name* string, *age* int, *pop* float)
Group (*gid* string, *name* string)
Member (*uid* int, *gid* string)

A more complicated example

- Names of all groups that Lisa and Alice are both in

```
SELECT g.name
        FROM User u1, User u2, Member m1, Member m2, Group g
        WHERE u1.name = 'Lisa' AND u2.name = 'Alice'
                AND u1.uid = m1.uid AND u2.uid=m2.uid
                AND m1.gid = g.gid AND m2.gid = g.gid;
```

User (*uid* int, *name* string, *age* int, *pop* float)
Group (*gid* string, *name* string)
Member (*uid* int, *gid* string)

Why SFW statements?

- Many queries can be written using only selection, projection, and cross product (or join)

- These queries can be written in a canonical form

$$\pi_L\left(\sigma_p(R_1 \times \cdots \times R_m)\right)$$

SELECT L
FROM $R_1, R_2, \ldots, R_m$
WHERE p

  - $\pi_{R.A,S.B}\left(R \bowtie_{p_1} S\right) \bowtie_{p_2} \left(\pi_{T.C}\sigma_{p_3} T\right)$
  but equivalent to $\pi_{R.A,S.B,T.C}\sigma_{p_1 \wedge p_2 \wedge p_3}(R \times S \times T)$ (why?)
  - Can be captured by SFW statements

Semantics of SFW

- SELECT $A_1, A_2, \ldots, A_n$
  FROM $R_1, R_2, \ldots, R_m$
  WHERE $condition;$
- For each t_1 in R_1:
  For each t_2 in R_2:
  ……
  For each t_m in R_m:
  If $condition$ is true over $t_1, t_2, \ldots, t_m$:
  Compute and output $A_1, A_2, \ldots, A_n$ as a row
- $t_1, t_2, \ldots, t_m$ are often called tuple variables

In class exercises

User (*uid* int, *name* string, *age* int, *pop* float)
Group (*gid* string, *name* string)
Member (*uid* int, *gid* string)

- List user names whose popularity is b/w 0.5 and 0.9

```
SELECT name FROM User WHERE pop > 0.5 AND pop < 0.9;
```

- List the group ids that a user with id 134 belongs to

```
SELECT gid FROM Member WHERE uid=134;
```

- List the group ids that Lisa belongs to

```
SELECT gid
        FROM Member m, User u
        WHERE u.name='Lisa' AND m.uid=u.uid;
```

In class exercises

User (*uid* int, *name* string, *age* int, *pop* float)
Group (*gid* string, *name* string)
Member (*uid* int, *gid* string)

- List the group names that Lisa belongs to

```
SELECT g.name
        FROM Member m, User u, Group g
        WHERE u.name='Lisa' AND m.uid=u.uid AND m.gid = g.gid;
```

- List user ids belonging to at least 2 groups

```
SELECT m1.uid
        FROM Member m1, Member m2
        WHERE m1.uid=m2.uid AND m1.gid < m2.gid;
```

a user joins three groups?

SQL features covered so far

- Basics
  - Data-definition language (DDL): define/modify schemas, delete relations
  - Data-manipulation language (DML): query data
    - SELECT-FROM-WHERE
    - Set v.s. Bag

Set versus bag

- Set versus Multi-set

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
…	…	…	…

$\pi_{age} User$

age
10
8
…

Set semantics

- No duplicates
- Relational algebra use set semantics

```
SELECT age
FROM User;
```

age
10
10
8
8
…

Bag semantics

- Duplicates allowed
- Rows in output = rows in input (w/o where clause)
- SQL uses bag semantics by default

A case for bag semantics

- Efficiency
  - Saves time of eliminating duplicates

$\pi_{age}User$

```
SELECT age
FROM User;
```

- Which one is more useful?
  - The first query just returns all possible user ages in the table
  - The second query returns the user age distribution

- Besides, SQL provides the option of set semantics with DISTINCT keyword

DISTINCT - Forcing set semantics

- IDs of all pairs of users that belong to one group

```
SELECT m1.uid AS uid1, m2.uid AS uid2
        FROM Member AS m1, Member AS m2
        WHERE m1.gid = m2.gid
                AND m1.uid < m2.uid;
```

→ Say Lisa and Alice are in both the book club and the student government, their id pairs will appear twice

- Remove duplicate (uid1, uid2) pairs from the output

```
SELECT DISTINCT m1.uid AS uid1, m2.uid AS uid2
        FROM Member AS m1, Member AS m2
        WHERE m1.gid = m2.gid;
                AND m1.uid < m2.uid;
```

Semantics of SFW with DISTINCT

- SELECT [DISTINCT] $A_1, A_2, \ldots, A_n$
  FROM $R_1, R_2, \ldots, R_m$
  WHERE $condition$;
- For each t_1 in R_1:
  For each t_2 in R_2:

  ……

  For each t_m in R_m:

  If $condition$ is true over $t_1, t_2, \ldots, t_m$:
  Compute and output $A_1, A_2, \ldots, A_n$ as a row

If DISTINCT is present
Eliminate duplicate rows in output

SQL set and bag operations

- Set: UNION, EXCEPT, INTERSECT
  - Exactly like set ∪, −, and ∩ in relational algebra
  - Duplicates in input tables, if any, are first eliminated
  - Duplicates in result are also eliminated

Bucket1

fruit
apple
apple
orange

Bucket2

fruit
apple
orange
orange

```
(SELECT * FROM Bucket1)
UNION
(SELECT * FROM Bucket2);
```

fruit
apple
orange

```
(SELECT * FROM Bucket1)
EXCEPT
(SELECT * FROM Bucket2);
```

fruit

```
(SELECT * FROM Bucket1)
INTERSECT
(SELECT * FROM Bucket2);
```

fruit
apple
orange

SQL set and bag operations

- Set: UNION, EXCEPT, INTERSECT
  - Exactly like set ∪, −, and ∩ in relational algebra
- Bag: UNION ALL, EXCEPT ALL, INTERSECT ALL
  - Think of each row as having an implicit count (the number of times it appears in the table)

Bucket1

fruit
apple
apple
orange

apple: 2
orange:1

Bucket2

fruit
apple
orange
orange

apple: 1
orange:2

```
(SELECT * FROM Bucket1)
UNION ALL
(SELECT * FROM Bucket2);
```

sum up the two counts

fruit
apple
apple
orange
apple
orange
orange

apple: 3
orange:3

SQL set and bag operations

- Set: UNION, EXCEPT, INTERSECT
  - Exactly like set ∪, −, and ∩ in relational algebra
- Bag: UNION ALL, EXCEPT ALL, INTERSECT ALL
  - Think of each row as having an implicit count (the number of times it appears in the table)

Bucket1

fruit
apple
apple
orange

apple: 2
orange:1

Bucket2

fruit
apple
orange
orange

apple: 1
orange:2

```
(SELECT * FROM Bucket1)
EXCEPT ALL
(SELECT * FROM Bucket2);
```

proper-subtract the two counts

fruit
apple

apple: 1
orange:0

SQL set and bag operations

- Set: UNION, EXCEPT, INTERSECT
  - Exactly like set ∪, −, and ∩ in relational algebra
- Bag: UNION ALL, EXCEPT ALL, INTERSECT ALL
  - Think of each row as having an implicit count (the number of times it appears in the table)

Bucket1

fruit
apple
apple
orange

apple: 2
orange:1

Bucket2

fruit
apple
orange
orange

apple: 1
orange:2

```
(SELECT * FROM Bucket1)
INTERSECT ALL
(SELECT * FROM Bucket2);
```

take the minimum of the two counts

fruit
apple
orange

apple: 1
orange:1

Set versus bag operations

Consider *Poke* (*uid1*, *uid2*, *timestamp*):

- uid1 poked uid2 at timestamp

How do these two queries differ?

```
Q1:
(SELECT uid1 FROM Poke)
EXCEPT
(SELECT uid2 FROM Poke);
```

Users who poked others but never got poked by others

```
Q2:
(SELECT uid1 FROM Poke)
EXCEPT ALL
(SELECT uid2 FROM Poke);
```

Users who poked others more than others poked them

In class exercises

- What is the output of these queries?

User

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

Member

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

How about UNION?

```
SELECT gid
FROM Member m, User u
WHERE u.name='Lisa' AND
u.uid=m.uid
```

```
SELECT gid
FROM Member m, User u
WHERE u.name='Lisa' AND u.uid=m.uid
UNION ALL
SELECT gid
FROM Member m, User u
WHERE u.name='Alice' AND u.uid=m.uid
```

SQL features covered so far

- Basics
  - Data-definition language (DDL): define/modify schemas, delete relations
  - Data-manipulation language (DML): query data
    - SELECT-FROM-WHERE
    - Set/bag (DISTINCT, UNION/EXCEPT/INTERSECT (ALL))
    - Nested SQL queries

Post-Lecture: Practice SQL queries

- School servers have db2 installed
  - Instructions in db2 tutorial posted as the supplementary materials with the project description
  - Instructions in Assignment 1

- The textbook's website has an SQLite db that runs in the browser: https://www.db-book.com/university-lab-dir/sqljs.html