Structured Query Language (SQL)

Yanlei Diao
UMass Amherst

Slides Courtesy of R. Ramakrishnan, J. Gehrke, and G. Miklau

Structured Query Language (SQL)

- Foundation
  - Semantics is based on relational calculus
  - Evaluation is based on relational algebra
  - Data model is a multiset model (extension of a set model)

1. Data Manipulation Language (DML)
   - posing queries
   - operating on tuples

2. Data Definition Language (DDL)
   - operating on tables/views

SQL Overview

- Query capabilities
  - SELECT-FROM-WHERE blocks
  - Set operations (union, intersect, except)
  - Nested queries (correlation)
  - Ordering
  - Aggregation and grouping
  - Null values
- Database updates
- Tables and views
- Integrity constraints

Example Instances

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

Basic SQL Query

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

- relation-list: a list of relation names, possibly each with a range-variable.
- qualification: predicates combined using AND, OR and NOT.
  - predicate: attr op const or attr1 op attr2, op: <, >, >=, <=, =, <>
- target-list: a list of attributes of relations in relation-list
  - DISTINCT indicates no duplicates in the answer. Default is that duplicates are not eliminated!
  - SQL uses a multiset-based model!

Conceptual Evaluation Strategy

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

- relation-list: cross-product (\( \times \))
- qualification: selection (\( \sigma \))
- target-list: projection (\( \pi \))
  - duplicate elimination if DISTINCT

- This is possibly the least efficient way to execute the query! Leave the issue to Query Optimizer…
An Example SQL Query

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

What is the relational algebra for this query?

\[ \pi_{\text{name}}((\sigma_{\text{bid}=103} \text{Reserves}) \bowtie \text{Sailors}) \]

A Note on Range Variables

- Really needed only if the same relation appears twice in the `FROM` clause.

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

OR

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

\[ \text{It is good style, however, to use range variables always!} \]

String Pattern Matching

- Find the ages of sailors whose names begin with ‘A’, end with ‘M’, and contain at least one character between ‘A’ and ‘M’.

```
SELECT S.age
FROM Sailors S
WHERE S.name LIKE ‘A%M’;
```

- ‘_’ stands for any single character.
- ‘%’ stands for 0 or more arbitrary characters.

Arithmetic Expressions

```
SELECT S.age1 = S.age-5, 2*S.age AS age2
FROM Sailors S
WHERE S.name LIKE ‘A%M’;
```

- Arithmetic expressions can also appear in the predicates in `WHERE`.

Q1: What Does the Query Compute?

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

- Q2: Would adding `DISTINCT` to this query make a difference?

- Q3: What if we replace `S.sid` by `S.name` in the `SELECT` clause and then add `DISTINCT`? Compare num of results with Q2.

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

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>rating</th>
<th>age</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7.0</td>
<td>45.0</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8.0</td>
<td>55.5</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10.0</td>
<td>35.0</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>103</td>
<td></td>
</tr>
</tbody>
</table>

Q2's answer:

- X

Q3's answer:

- X

A Note on Range Variables

- Really needed only if the same relation appears twice in the `FROM` clause.
### SQL Overview

- **Query capabilities**
  - SELECT-FROM-WHERE blocks
  - Set operations (union, intersect, except)
  - Nested queries (correlation)
  - Ordering
  - Aggregation & Grouping
  - Null values
- **Database updates**
- **Tables and views**
- **Integrity constraints**

### Find sid’s of sailors who’ve reserved a red or a green boat

- If we replace OR by AND in this query, what do we get?
  ```sql
  SELECT DISTINCT R.sid
  FROM Reserves R, Boats B
  WHERE R.bid=B.bid
  AND (B.color='red' OR B.color='green');
  ```

### Find sid’s of sailors who’ve reserved a red and a green boat

- UNION: computes the union of any two union-compatible sets of tuples (which are themselves the result of SQL queries).
  - Duplicates after UNION?
  - What if we remove the DISTINCT keyword?

### Find sid’s of sailors who’ve reserved a red and a green boat

- INTERSECT: computes the intersection of any two union-compatible sets of tuples.
  - Duplicates after INTERSECT?
  - What if we remove the DISTINCT keyword?

### Find sid’s of sailors who’ve reserved a red and a green boat

- INTERSECT is only a derived operator, we can rewrite it:
  ```sql
  SELECT DISTINCT R1.sid
  FROM Reserves R1, Boats B1,
  Reserves R2, Boats B2
  WHERE R1.bid=B1.bid
  AND B1.color='red'
  INTERSECT
  SELECT DISTINCT R2.sid
  FROM Reserves R2, Boats B2
  WHERE R2.bid=B2.bid
  AND B2.color='green';
  ```

### SQL Overview

- **Query capabilities**
  - SELECT-FROM-WHERE blocks
  - Set operations (union, intersect, except)
  - Nested queries (correlation)
  - Ordering
  - Aggregation & Grouping
  - Null values
- **Database updates**
- **Tables and views**
- **Integrity constraints**
Nested Queries

- A nested query has another query embedded within it.
- The embedded query is called the subquery.
- The outer one is called the outer query.

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

- The subquery often appears in the WHERE clause:

- A subquery can also appear in the FROM clause. An example is shown later.

Set Comparison Operators in WHERE

- Set comparisons:
  - `attr IN R` — true if \( R \) contains `attr`
  - `EXISTS R` — true if \( R \) is non-empty
  - `UNIQUE R` — true if no duplicates in \( R \)
  - Any of the above comparators with a proceeding `NOT`

- Set comparisons using a comparator `op {<,<=,=,<>, >=,}>`:
  - `attr op ALL R` — every element of \( R \) satisfies condition
  - `attr op ANY R` — some element of \( R \) satisfies condition

```
'Sid IN R'  equivalent to  '"attr = ANY R'  
'Sid NOT IN R'  equivalent to  '"attr <> ALL R'  
```

Find sid’s of sailors who’ve reserved a red and a green boat

```
INTERSECT computes the intersection of any two union-compatible sets of tuples.
```

Simulating INTERSECT

```
SELECT DISTINCT R.sid
FROM    Reserves R, Boats B
WHERE  R.bid=B.bid
AND B.color='red'
AND R.sid IN ( SELECT DISTINCT R.sid
                      FROM    Reserves R, Boats B
                      WHERE  R.bid=B.bid
                      AND B.color='green' );
```

Find sid’s of sailors who’ve reserved a green boat but not a red boat

```
EXCEPT computes set difference
```

```
SELECT DISTINCT S.sid
FROM    Reserves R, Boats B
WHERE  R.bid=R.bid
AND B.color='green'
EXCEPT
SELECT DISTINCT S.sid
FROM    Reserves R, Boats B
WHERE  R.bid=R.bid
AND B.color='red';
```
Simulating EXCEPT (set difference)

```sql
SELECT DISTINCT R.sid
FROM Reserves R, Boats B
WHERE R.bid=B.bid
AND B.color = 'green'
AND R.sid NOT IN ( 
  SELECT DISTINCT R.sid
  FROM Reserves R, Boats B
  WHERE R.bid=B.bid
  AND B.color = 'red'
);```

Finding Extreme Values

- Find the sailors with the highest rating

```sql
SELECT S.sid
FROM Sailors S
WHERE S.rating >= ALL (SELECT S2.rating
                        FROM Sailors S2)
```

Correlated Subqueries

- A subquery that depends on tables mentioned in the outer query is a correlated subquery.
- In conceptual evaluation, must re-compute subquery for each row of the outer query.

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

Find the names of sailors who’ve reserved all boats

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

SQL Overview

- Query capabilities
  - SELECT-FROM-WHERE blocks
  - Set operations (union, intersect, except)
  - Nested queries (correlation)
- Ordering
  - Aggregation & Grouping
  - Null values
- Database updates
- Tables and views
- Integrity constraints
ORDER BY

- Return the name and age of sailors rated level 8 or above in increasing (decreasing) order of age.

```sql
SELECT S.sname, S.age
FROM Sailors S
WHERE S.rating > 8
ORDER BY S.age [ASC | DESC];
```

TOP-K Queries

- Return the name and age of the ten youngest sailors rated level 8 or above.

```sql
SELECT S.sname, S.age
FROM Sailors S
WHERE S.rating >= 8
ORDER BY S.age ASC
LIMIT 10;
```

SQL Overview

- Query capabilities
  - SELECT-FROM-WHERE blocks
  - Set operations (union, intersect, except)
  - Nested queries (correlation)
  - Ordering
    - Aggregation and grouping
  - Null values
  - Database updates
  - Tables and views
  - Integrity constraints

Aggregate Operators

- Aggregate functions take a relation (single column or multiple columns), and return a value.

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

Example Aggregate Operators

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

Aggregate Operators

- Take a relation (single column or multiple columns), return a value.
- Significant extension of original relational algebra.

```sql
COUNT(*)
COUNT (DISTINCT A)
SUM (DISTINCT A)
AVG (DISTINCT A)
MAX (A)
MIN (A)
```
Find name and age of the oldest sailor(s)

- The first query is illegal! (We’ll look into the reason more when we discuss GROUP BY.)

```
SELECT S.sname, MAX(S.age)
FROM Sailors S;
```

```
SELECT S.sname, S.age
FROM Sailors S
WHERE S.age >= ALL
(SELECT S2.age
FROM Sailors S2);
```

Queries With GROUP BY and HAVING

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

- A group is a set of tuples that have the same value for all attributes in grouping-list.
- Query returns a single answer tuple for each group!
- The target-list can only contain:
  (i) attributes that have a single value for a group (e.g., S.rating), or
  (ii) aggregate operations on other attributes, e.g., MIN(S.age).

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

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

Conceptual Evaluation, extended

- The cross-product of relation-list is computed.
- Tuples that fail qualification are discarded.
- The remaining tuples are partitioned into groups by the value of attributes in grouping-list.
- The group-qualification, if present, eliminates some groups.
  - Group-qualification must have a single value per group!
  - A single answer tuple is produced for each qualifying group.

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

```
<table>
<thead>
<tr>
<th>rating</th>
<th>age</th>
<th>min_age</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>45.0</td>
<td>45.0</td>
</tr>
<tr>
<td>1</td>
<td>33.0</td>
<td>33.0</td>
</tr>
<tr>
<td>8</td>
<td>55.5</td>
<td>33.0</td>
</tr>
<tr>
<td>10</td>
<td>35.0</td>
<td>45.0</td>
</tr>
<tr>
<td>7</td>
<td>35.0</td>
<td>35.0</td>
</tr>
<tr>
<td>10</td>
<td>35.0</td>
<td>35.0</td>
</tr>
<tr>
<td>9</td>
<td>35.0</td>
<td>35.0</td>
</tr>
<tr>
<td>3</td>
<td>25.5</td>
<td>35.0</td>
</tr>
<tr>
<td>3</td>
<td>25.5</td>
<td>35.0</td>
</tr>
</tbody>
</table>
```

Motivation for Grouping

- What if we want to apply aggregate operators to each group (subset) of tuples?
- Find the age of the youngest sailor for each rating level.
  - If we know that rating values ∈ [1, 10], write 10 queries like:
    ```
    SELECT MIN(S.age)
    FROM Sailors S
    WHERE Rating = i
    ```
  - In general, we don’t know how many rating levels exist, and what the rating values for these levels are!
What Does the Query Compute?

```sql
SELECT Temp.rating, Temp.avgage
FROM (SELECT S.rating, AVG(S.age) AS avgage
      FROM Sailors S
      GROUP BY S.rating)
      AS Temp
WHERE Temp.avgage = (SELECT MIN (Temp.avgage)
                      FROM Temp);
```

- Derived table: result of an SQL query as input to the FROM clause of another query
- Computed once before the other query is evaluated.

SQL Overview

- Query capabilities
  - SELECT-FROM-WHERE blocks
  - Set operations (union, intersect, except)
  - Nested queries (correlation)
  - Aggregation & Grouping
  - Ordering
- Derived table
- Database updates
- Tables and views
- Integrity constraints

NULLS in SQL

- Whenever we don’t have a value, put a NULL
  - Value does not exist.
  - Value exists but is unknown.
  - This attribute is not applicable.
- The schema specifies for each attribute whether it can be null (e.g., NOT NULL).
- How does SQL cope with tables that have NULLs?

NULL Values

- If x = NULL, then 4*(3-x)/7 is still NULL
- If x = NULL, then x = “Joe” is UNKNOWN
- In SQL there are three boolean values:
  - FALSE = 0
  - UNKNOWN = 0.5
  - TRUE = 1

Boolean Expressions involving NULL

- C1 AND C2 = min(C1, C2)
- C1 OR C2 = max(C1, C2)
- NOT C1 = 1 – C1
- Rule in SQL: include only tuples that yield TRUE

Null Values

- Unexpected behavior:

```sql
SELECT *
FROM Person
WHERE (age < 25) AND (height > 6 OR weight > 190);
```

- age = 20
- height = NULL
- weight = 200

John’s age is NULL
**Null Values**

- Can test for NULL explicitly:
  - x IS NULL
  - x IS NOT NULL

```
SELECT *
FROM  Person
WHERE age < 25 OR age >= 25 OR age IS NULL;
```

- Now it includes all Persons.

---

**SQL Overview**

- Query capabilities
  - SELECT-FROM-WHERE blocks
  - Set operations (union, intersect, except)
  - Nested queries (correlation)
  - Aggregation & Grouping
  - Ordering
  - Null values

- Database updates
- Tables and views
- Integrity constraints

---

**Modifying the Database**

Three kinds of modifications:
- Insert - create new tuple(s)
- Delete - remove existing tuple(s)
- Update - modify existing tuple(s)

- Sometimes they are all called "updates".

---

**Insertions**

General form:

```
INSERT INTO R(A1, ..., An)
VALUES (v1, ..., vn);
```

Example: Insert a new sailor to the database:

```
INSERT INTO Sailors(sid, sname, rating, age)
VALUES (3212, 'Fred', 9, 44);
```

Can omit attributes; a missing attribute is NULL. May drop attribute names if give values of all attributes in order.

---

**Deletions**

Example: delete all tuples that satisfy a condition

```
DELETE FROM Sailors
WHERE S.sname = 'Harry';
```

Fact about SQL: there is no way to delete only a single occurrence of a tuple that appears twice in a relation.

---

Example: Insert multiple tuples to Sailors:

```
INSERT INTO Sailors(sid, sname)
SELECT DISTINCT B.sid, B.name
FROM  Boaters B
WHERE  Boaters.rank = 'captain';
```

The query replaces the VALUES keyword.
Updates

Examples:

- UPDATE Employees
  SET salary = salary * 1.1;

- UPDATE Sailors S
  SET S.rating = S.rating + 1
  WHERE S.sid IN
  (SELECT sid
   FROM Reserves R
   WHERE R.date = 'Oct, 25');

SQL Overview

- Query capabilities
  - SELECT-FROM-WHERE blocks
  - Set operations (union, intersect, except)
  - Nested queries (correlation)
  - Aggregation & Grouping
  - Ordering
  - Null values
- Database updates
- Tables and views
- Integrity constraints

Creating Tables

- CREATE TABLE Sailors
  (sid INTEGER,
   sname VARCHAR(50) NOT NULL,
   rating INTEGER,
   age REAL,
   PRIMARY KEY (sid));

- CREATE TABLE Reserves
  (sid INTEGER,
   bid INTEGER,
   day DATE,
   PRIMARY KEY (sid,bid,day),
   FOREIGN KEY (sid) REFERENCES Sailors (sid)
    ON DELETE NO ACTION ON UPDATE CASCADE,
   FOREIGN KEY (bid) REFERENCES Boats (bid)
    ON DELETE SET DEFAULT ON UPDATE CASCADE);

- CREATE TABLE Boats
  (bid INTEGER,
   bname CHAR(20),
   color CHAR(20),
   PRIMARY KEY (bid)
    UNIQUE (bname));

Destroying and Altering Tables

- DROP TABLE Sailors;
- ALTER TABLE Sailors
  ADD COLUMN credit_card:CHAR(40);

- The schema is altered by adding a new field; every tuple in the current instance is extended with a null value in the new field.

Views

- A view is like a relation, but we store a definition, rather than a set of tuples.

- CREATE VIEW RedBoatLovers (sid, name, bid)
  AS SELECT S.sid, S.sname, B.bid
  FROM Sailors S, Reserves R, Boats B
  WHERE S.sid = R.sid and R.bid = B.bid
    and B.color = 'red';

- Views can be dropped using DROP VIEW command.
  - DROP TABLE if there’s a view on the table? Options available...

Uses of Views

- Security/Privacy
  - Views can be used to present necessary information (or a summary) while hiding details in underlying relation(s).
  - E.g., hiding sailors’ credit card from the boat repair dept.
- Logical data independence
  - User application defined on a view is unchanged when underlying table changes.
- Computational benefits
  - Result of a complex query is frequently used; define a view and materialize the result.
  - Online Analytical Processing (OLAP)
SQL Overview

- Query capabilities
  - SELECT-FROM-WHERE blocks
  - Set operations (union, intersect, except)
  - Nested queries (correlation)
  - Aggregation and grouping
  - Ordering
  - Null values
- Database updates
- Tables and views
  - Integrity constraints

Integrity Constraints (Review)

- Types of integrity constraints in SQL:
  - Attribute constraints: domain, NOT NULL
  - Key constraints: PRIMARY KEY, UNIQUE
  - Foreign key constraints: FOREIGN KEY
  - General constraints: CHECK, ASSERTION
- Inserts/deletes/updates that violate IC’s are disallowed.

General Constraints

- Two forms: CHECK (single table constraint) and ASSERTION (multiple-table constraint).

```
CREATE TABLE Sailors
( sid INTEGER,
  sname CHAR(50),
  rating INTEGER,
  age REAL,
  PRIMARY KEY (sid),
  CHECK (rating >= 1 AND rating <= 10));
```

Constraints over Multiple Relations

- Number of boats plus number of sailors is < 100:

```
CREATE ASSERTION smallClub
CHECK ((SELECT COUNT (S.sid) FROM Sailors S) +
       (SELECT COUNT (B.bid) FROM Boats B) < 100);
```

- ASSERTION is a constraint over both tables; checked whenever one of the table is modified.

Review: Participation Constraints

- Participation constraint:
  - Every employee works in at least one dept.
  - Each Dept has at least one employee.
- Participation + Key constraints:
  - Every department must have one manager.

Assertion for Participation Constraint

```
CREATE ASSERTION Participation_Employee
CHECK ((SELECT COUNT (DISTINCT ssn) FROM Employees) =
       (SELECT COUNT (DISTINCT ssn) FROM Works_In));
```

```
CREATE ASSERTION Participation_Department
CHECK ((SELECT COUNT (DISTINCT did) FROM Departments) =
       (SELECT COUNT (DISTINCT did) FROM Works_In));
```
Using Triggers Instead

- A **trigger** is associated with a table, and activates when an event (insert, delete, update) occurs to a table
- Used to check values or perform computation

```sql
mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
Query OK, 0 rows affected (0.03 sec)
```

```sql
mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account
   -> FOR EACH ROW SET @sum = @sum + NEW.amount;
Query OK, 0 rows affected (0.06 sec)
```

- For full details, see: