CompSci 516
Database Systems

Lecture 3
SQL
RC
RA

Instructor: Sudeepa Roy

Announcements

- Lab-1 makeup instructions sent on piazza
- Let me know if you are still not on piazza
- HW1 will be posted after the class
  - Deadlines in stages
  - First deadline on 09/17

Today's topic

- Finish SQL
- RC

- Next week:
  - Tuesday: Guest Lecture by Junyang Gao: RA
  - Thursday: Lab on RA

Acknowledgement:
The following slides have been created adapting the instructor material of the [RG] book provided by the authors Dr. Ramakrishnan and Dr. Gehrke.

Nulls and Views in SQL

Null Values

- Field values in a tuple are sometimes
  - unknown, e.g., a rating has not been assigned, or
  - inapplicable, e.g., no spouse's name
  - SQL provides a special value null for such situations.

Standard Boolean 2-valued logic

- True = 1, False = 0
- Suppose X = 5
  - (X < 100) AND (X >= 1) is T ∧ T = T
  - (X > 100) OR (X >= 1) is F ∨ T = T
  - (X > 100) AND (X >= 1) is F ∧ T = F
  - NOT(X = 5) is ¬T = F

- Intuitively,
  - T = 1, F = 0
  - For V1, V2 ∈ {1, 0}
  - V1 ∧ V2 = MIN (V1, V2)
  - V1 ∨ V2 = MAX(V1, V2)
  - ¬(V1) = 1 – V1

2-valued logic does not work for nulls

- Suppose rating = null, X = 5
- Is rating>8 true or false?
- What about AND, OR and NOT connectives?
  - (rating > 8) AND (X = 5)?
- What if we have such a condition in the WHERE clause?

3-Valued Logic For Null

End of Lecture 3

- TRUE (= 1), FALSE (= 0), UNKNOWN (= 0.5)
  - unknown is treated as 0.5
- Now you can apply rules from 2-valued logic!
  - For V1, V2 ∈ {1, 0, 0.5}
  - V1 ∧ V2 = MIN (V1, V2)
  - V1 ∨ V2 = MAX(V1, V2)
  - ¬(V1) = 1 – V1
- Therefore,
  - NOT UNKNOWN = UNKNOWN
  - UNKNOWN OR TRUE = TRUE
  - UNKNOWN AND TRUE = UNKNOWN
  - UNKNOWN AND FALSE = FALSE
  - UNKNOWN OR FALSE = UNKNOWN

New issues for Null

- The presence of null complicates many issues. E.g.:
  - Special operators needed to check if value IS/IS NOT NULL
  - Be careful!
  - "WHERE X = NULL" does not work!
  - Need to write "WHERE X IS NULL"
- Meaning of constructs must be defined carefully
  - e.g., WHERE clause eliminates rows that don't evaluate to true
  - So not only FALSE, but UNKNOWNs are eliminated too
  - very important to remember!
- But NULL allows new operators (e.g. outer joins)
- Can force "no nulls" while creating a table
  - sname char(20) NOT NULL
  - primary key is always not null

Aggregates with NULL

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

R1

- What do you get for
- SELECT count(*) from R1?
- SELECT count(rating) from R1?

Aggregates with NULL

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

R1

- What do you get for
- SELECT count(*) from R1?
- SELECT count(rating) from R1?

Aggregates with NULL

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

R1

- What do you get for
- SELECT count(*) from R1?
- SELECT count(rating) from R1?

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

R2

- What do you get for
- SELECT count(*) from R2?
- SELECT count(rating) from R2?

Aggregates with NULL

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

R1

- What do you get for
- SELECT count(*) from R1?
- SELECT count(rating) from R1?

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

R2

- What do you get for
- SELECT count(*) from R2?
- SELECT count(rating) from R2?

Aggregates with NULL

- COUNT, SUM, AVG, MIN, MAX (with or without DISTINCT)
  - Discards null values first
  - Then applies the aggregate
  - Except count(*)
- If only applied to null values, the result is null

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

R2

- SELECT sum(rating) from R2?

sid	sname	rating	age
22	dustin	null	45
31	lubber	null	55
58	rusty	null	35

R3

- SELECT sum(rating) from R3?

Creating Relations in SQL

- Creates the "Students" relation
  - the type (domain) of each field is specified
  - enforced by the DBMS whenever tuples are added or modified

```
CREATE TABLE Students
  (sid CHAR(20),
   name CHAR(20),
   login CHAR(10),
   age INTEGER,
   gpa REAL)
```

- As another example, the "Enrolled" table holds information about courses that students take

```
CREATE TABLE Enrolled
  (sid CHAR(20),
   cid CHAR(20),
   grade CHAR(2))
```

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Students

sid	cid	grade
53831	Carnatic101	C
53831	Reggae203	B
53650	Topology112	A
53666	History105	B

Enrolled

Destroying and Altering Relations

```
DROP TABLE Students
```

- Destroys the relation Students
  - The schema information *and* the tuples are deleted.

```
ALTER TABLE Students
  ADD COLUMN firstYear: integer
```

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

Adding and Deleting Tuples

- Can insert a single tuple using:

```
INSERT INTO Students (sid, name, login, age, gpa)
VALUES (53688, 'Smith', 'smith@ee', 18, 3.2)
```

- Can delete all tuples satisfying some condition (e.g., name = Smith):

```
DELETE
FROM Students S
WHERE S.name = 'Smith'
```


Integrity Constraints (ICs)

- IC: condition that must be true for any instance of the database
  - e.g., domain constraints
  - ICs are specified when schema is defined
  - ICs are checked when relations are modified
- A legal instance of a relation is one that satisfies all specified ICs
  - DBMS will not allow illegal instances
- If the DBMS checks ICs, stored data is more faithful to real-world meaning
  - Avoids data entry errors, too!

Keys in a Database

- Key / Candidate Key
- Primary Key
- Super Key
- Foreign Key

- Primary key attributes are underlined in a schema
  - Person(pid, address, name)
  - Person2(address, name, age, job)

Primary Key Constraints

- A set of fields is a key for a relation if :
  1. No two distinct tuples can have same values in all key fields, and
  2. This is not true for any subset of the key
- Part 2 false? A superkey
- If there are > 1 keys for a relation, one of the keys is chosen (by DBA = DB admin) to be the primary key
  - E.g., sid is a key for Students
  - The set {sid, gpa} is a superkey.
- Any possible benefit to refer to a tuple using primary key (than any key)?

Primary and Candidate Keys in SQL

- Possibly many candidate keys
  - specified using UNIQUE
  - one of which is chosen as the primary key.

- "For a given student and course, there is a single grade."

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  ???)
```


Primary and Candidate Keys in SQL

- Possibly many candidate keys
  - specified using UNIQUE
  - one of which is chosen as the primary key.

- "For a given student and course, there is a single grade."

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  (sid, cid) )
```


Primary and Candidate Keys in SQL

- Possibly many candidate keys
  - specified using UNIQUE
  - one of which is chosen as the primary key.

- "For a given student and course, there is a single grade."
- vs.
- "Students can take only one course, and receive a single grade for that course; further, no two students in a course receive the same grade."

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  (sid, cid) )
```

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY ???,
   UNIQUE ??? )
```


Primary and Candidate Keys in SQL

- Possibly many candidate keys
  - specified using UNIQUE
  - one of which is chosen as the primary key.

- "For a given student and course, there is a single grade."
- vs.
- "Students can take only one course, and receive a single grade for that course; further, no two students in a course receive the same grade."

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  (sid,cid) )
```

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   ???
```

Primary and Candidate Keys in SQL

- Possibly many candidate keys
  - specified using UNIQUE
  - one of which is chosen as the primary key.

- "For a given student and course, there is a single grade."
- vs.
- "Students can take only one course, and receive a single grade for that course; further, no two students in a course receive the same grade."
- Used carelessly, an IC can prevent the storage of database instances that arise in practice!

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  (sid,cid) )
```

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   ????
```


Foreign Keys, Referential Integrity

- Foreign key : Set of fields in one relation that is used to `refer' to a tuple in another relation
  - Must correspond to primary key of the second relation
  - Like a `logical pointer'
- E.g. sid is a foreign key referring to Students:
  - Enrolled(sid: string, cid: string, grade: string)
  - If all foreign key constraints are enforced, referential integrity is achieved
  - i.e., no dangling references

Foreign Keys in SQL

- Only students listed in the Students relation should be allowed to enroll for courses

```
CREATE TABLE Enrolled
  (sid CHAR(20), cid CHAR(20), grade CHAR(2),
   PRIMARY KEY  (sid,cid),
   FOREIGN KEY (sid) REFERENCES Students )
```

Enrolled

sid	cid	grade
53666	Carnatic101	C
53666	Reggae203	B
53650	Topology112	A
53666	History105	B

Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Enforcing Referential Integrity

- Consider Students and Enrolled
  - sid in Enrolled is a foreign key that references Students.
- What should be done if an Enrolled tuple with a non-existent student id is inserted?
  - Reject it!
- What should be done if a Students tuple is deleted?
  - Three semantics allowed by SQL
  1. Also delete all Enrolled tuples that refer to it (cascade delete)
  2. Disallow deletion of a Students tuple that is referred to
  3. Set sid in Enrolled tuples that refer to it to a default sid
  4. (in addition in SQL): Set sid in Enrolled tuples that refer to it to a special value null, denoting `unknown' or `inapplicable'
- Similar if primary key of Students tuple is updated

Referential Integrity in SQL

- SQL/92 and SQL:1999 support all 4 options on deletes and updates.
  - Default is NO ACTION (delete/update is rejected)
  - CASCADE (also delete all tuples that refer to deleted tuple)
  - SET NULL / SET DEFAULT (sets foreign key value of referencing tuple)

```
CREATE TABLE Enrolled
  (sid CHAR(20) DEFAULT '000',
   cid CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  (sid,cid),
   FOREIGN KEY (sid)
    REFERENCES Students
     ON DELETE CASCADE
     ON UPDATE SET DEFAULT )
```


Where do ICs Come From?

- ICs are based upon the semantics of the real-world enterprise that is being described in the database relations
- Can we infer ICs from an instance?
  - We can check a database instance to see if an IC is violated, but we can NEVER infer that an IC is true by looking at an instance.
  - An IC is a statement about all possible instances!
  - From example, we know name is not a key, but the assertion that sid is a key is given to us.
- Key and foreign key ICs are the most common; more general ICs supported too

Example Instances

- What does the key (sid, bid, day) in Reserves mean?
- If the key for the Reserves relation contained only the attributes (*sid, bid*), how would the semantics differ?

Sailor

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

Reserves

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

Views

Students(sid, name)
Enrolled(sid, cid, grade)

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

```
CREATE VIEW YoungActiveStudents (name, grade)
    AS SELECT S.name, E.grade
    FROM Students S, Enrolled E
    WHERE S.sid = E.sid and S.age<21
```

- Views can be dropped using the DROP VIEW command
- Views and Security: Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s)
  - the above view hides courses "cid" from E

Can create a new table from a query on other tables too

SELECT… INTO…. FROM…. WHERE

```
SELECT S.name, E.grade
INTO YoungActiveStudents
FROM Students S, Enrolled E
WHERE S.sid = E.sid and S.age<21
```


"WITH" clause – very useful!

- You will find "WITH" clause very useful!

```
WITH Temp1 AS
    (SELECT ….. ..),
    Temp2 AS
    (SELECT ….. ..)
SELECT X, Y
FROM TEMP1, TEMP2
WHERE….
```

- Can simplify complex nested queries

Overview: General Constraints

- Useful when more general ICs than keys are involved
- There are also ASSERTIONS to specify constraints that span across multiple tables
- There are TRIGGERS too : procedure that starts automatically if specified changes occur to the DBMS

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

```
CREATE TABLE Reserves
    ( sname CHAR(10),
    bid INTEGER,
    day DATE,
    PRIMARY KEY (bid,day),
    CONSTRAINT noInterlakeRes
    CHECK (`Interlake' <>
        ( SELECT B.bname
        FROM Boats B
        WHERE B.bid=bid)))
```


Summary: SQL

- SQL has a huge number of constructs and possibilities
  - You need to learn and practice it on your own
- Can limit answers using "LIMIT" or "TOP" clauses
  - e.g. to output TOP 20 results according to an aggregate
  - also can sort using ASC or DESC keywords
- We learnt
  - Creating/modifying relations
  - Specifying integrity constraints
  - Key/candidate key, superkey, primary key, foreign key
  - Conceptual evaluation of SQL queries
  - Joins
  - Group bys and aggregates
  - Nested queries
  - NULLs
  - Views

Relational Query Languages

Relational Query Languages

- Query languages: Allow manipulation and retrieval of data from a database
- Relational model supports simple, powerful QLs:
  - Strong formal foundation based on logic
  - Allows for much optimization
- Query Languages != programming languages
  - QLs not intended to be used for complex calculations
  - QLs support easy, efficient access to large data sets

Formal Relational Query Languages

- Two "mathematical" Query Languages form the basis for "real" languages (e.g. SQL), and for implementation:
  - Relational Calculus: Lets users describe what they want, rather than how to compute it (Non-operational, declarative, or procedural)
  - Relational Algebra: More operational, very useful for representing execution plans
- Note: Declarative (RC, SQL) vs. Operational (RA)

Relational Calculus (RC)

Logic Notations

- ∃ There exists
- ∀ For all
- ∧ Logical AND
- ∨ Logical OR
- ¬ NOT
- ⇒ Implies

TRC: example

Sailors(sid, sname, rating, age)
Boats(bid, bname, color)
Reserves(sid, bid, day)

- Find the name and age of all sailors with a rating above 7

∃ There exists

{P | ∃ S ∈ Sailors (S.rating > 7 ∧ P.sname = S.sname ∧ P.age = S.age)}

- P is a tuple variable
  - with exactly two fields sname and age (schema of the output relation)
  - P.sname = S.sname ∧ P.age = S.age gives values to the fields of an answer tuple
- Use parentheses, ∀ ∃ ∨ ∧ > < = ≠ ¬ etc as necessary
- A ⇒ B is very useful too
  - next slide

A ⇒ B

- A "implies" B
- Equivalently, if A is true, B must be true
- Equivalently, ¬ A ∨ B, i.e.
  - either A is false (then B can be anything)
  - otherwise (i.e. A is true) B must be true

Useful Logical Equivalences

- ∀x P(x) = ¬∃x [¬P(x)]

∃	There exists
∀	For all
∧	Logical AND
∨	Logical OR
¬	NOT

- ¬(P∨Q) = ¬ P∧ ¬ Q
- ¬(P ∧ Q) = ¬ P ∨ ¬ Q

} de Morgan's laws

  – Similarly, ¬(¬P∨Q) = P∧ ¬ Q etc.

- A ⇒ B = ¬ A ∨ B

TRC: example

Sailors(sid, sname, rating, age)
Boats(bid, bname, color)
Reserves(sid, bid, day)

- Find the names of sailors who have reserved at least two boats

TRC: example

Sailors(sid, sname, rating, age)
Boats(bid, bname, color)
Reserves(sid, bid, day)

- Find the names of sailors who have reserved at least two boats

{P | ∃ S ∈ Sailors (∃ R1 ∈ Reserves ∃ R2 ∈ Reserves (S.sid = R1.sid ∧ S.sid = R2.sid ∧ R1.bid ≠ R2.bid) ∧ P.sname = S.sname)}

TRC: example

Sailors(sid, sname, rating, age)
Boats(bid, bname, color)
Reserves(sid, bid, day)

- Find the names of sailors who have reserved all boats
- Called the "Division" operation

TRC: example

Sailors(sid, sname, rating, age)
Boats(bid, bname, color)
Reserves(sid, bid, day)

- Find the names of sailors who have reserved all boats
- Division operation in RA!

{P | ∃ S ∈ Sailors [∀B ∈ Boats (∃ R ∈ Reserves (S.sid = R.sid ∧ R.bid = B.bid))] ∧ (P.sname = S.sname)}

TRC: example

Sailors(sid, sname, rating, age)
Boats(bid, bname, color)
Reserves(sid, bid, day)

- Find the names of sailors who have reserved all red boats

How will you change the previous TRC expression?

TRC: example

Sailors(sid, sname, rating, age)
Boats(bid, bname, color)
Reserves(sid, bid, day)

- Find the names of sailors who have reserved all red boats

{P | ∃ S ∈ Sailors (∀B ∈ Boats (B.color = 'red' ⇒ (∃ R ∈ Reserves (S.sid = R.sid ∧ R.bid = B.bid))) ∧ P.sname = S.sname)}

Recall that A ⇒ B is logically equivalent to ¬ A ∨ B

so ⇒ can be avoided, but it is cleaner and more intuitive

More Examples: RC

- The famous "Drinker-Beer-Bar" example!

UNDERSTAND THE DIFFERENCE IN ANSWERS
FOR ALL FOUR DRINKERS

Acknowledgement: examples and slides by Profs. Balazinska and Suciu, and the [GUW] book

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Drinker Category 1

Find drinkers that frequent some bar that serves some beer they like.

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Drinker Category 1

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Drinker Category 2

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Drinker Category 2

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Drinker Category 3

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Drinker Category 3

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Drinker Category 4

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Drinker Category 4

Why should we care about RC

- RC is declarative, like SQL, and unlike RA (which is operational)
- Gives foundation of database queries in first-order logic
  - you cannot express all aggregates in RC, e.g. cardinality of a relation or sum (possible in extended RA and SQL)
  - still can express conditions like "at least two tuples" (or any constant)
- RC expression may be much simpler than SQL queries
  - and easier to check for correctness than SQL
  - power to use $\forall$ and $\Rightarrow$
  - then you can systematically go to a "correct" SQL query

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

From RC to SQL

Query: Find drinkers that like some beer (so much) that they frequent all bars that serve it

$Q(x) = \exists y.\ Likes(x, y) \wedge \forall z.(Serves(z,y) \Rightarrow Frequents(x,z))$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

From RC to SQL

Query: Find drinkers that like some beer so much that they frequent all bars that serve it

$Q(x) = \exists y.\ Likes(x, y) \wedge \forall z.(Serves(z,y) \Rightarrow Frequents(x,z))$

$\equiv \quad Q(x) = \exists y.\ Likes(x, y) \wedge \forall z.(\neg Serves(z,y) \vee Frequents(x,z))$

Step 1: Replace $\forall$ with $\exists$ using de Morgan's Laws

$\forall x\ P(x)$ same as $\neg\exists x\ \neg P(x)$

$Q(x) = \exists y.\ Likes(x, y) \wedge \neg\exists z.(Serves(z,y) \wedge \neg Frequents(x,z))$

$\neg(\neg P \vee Q)$ same as $P \wedge \neg Q$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

From RC to SQL

Query: Find drinkers that like some beer so much that they frequent all bars that serve it

$Q(x) = \exists y.\ Likes(x, y) \wedge \neg \exists z.(Serves(z,y) \wedge \neg Frequents(x,z))$

Step 2: Translate into SQL

```
SELECT DISTINCT L.drinker
FROM Likes L
WHERE not exists
  (SELECT S.bar
   FROM Serves S
   WHERE L.beer=S.beer
      AND not exists (SELECT *
                      FROM Frequents F
                      WHERE F.drinker=L.drinker
                        AND F.bar=S.bar))
```

We will see a "methodical and correct" translation trough "safe queries" in Datalog