Querying Relational Data, SQL

CMPSCI 445

Fall 2008
Today

• Review of Joins
• Conclude relational algebra
• Begin SQL
Joins are the most common way to combine information from two tables.

- **Theta Join:**
  \[
  R_1 \Join_\theta R_2 = \sigma_\theta (R_1 \times R_2)
  \]

- **Equijoin:**
  \[
  R_1 \Join_{A=B} R_2 = \sigma_{A=B} (R_1 \times R_2)
  \]

- **Natural Join**
  \[
  R_1 \Join R_2 = \Pi_A(\sigma_C(R_1 \times R_2))
  \]
**Example Database**

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>Takes</th>
<th>COURSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>sid</td>
<td>cid</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>445</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>483</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>435</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PROFESSOR</th>
<th>Teaches</th>
</tr>
</thead>
<tbody>
<tr>
<td>fid</td>
<td>fid</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>
Natural join questions

• Given the schemas R(A, B, C, D), S(A, C, E), what is the schema of \( R \bowtie S \)?
  – \( R(A,B,C,D,E) \)

• Given \( R(A, B, C) \), \( S(D, E) \), what is \( R \bowtie S \)?
  – Cartesian Product

• Given \( R(A, B) \), \( S(A, B) \), what is \( R \bowtie S \)?
  – Intersection
Algebraic Equivalences

• Relational algebra has laws of commutativity, associativity, etc. that imply certain expressions are equivalent.

\[ \sigma_{c \land d}(R) \equiv \sigma_c(\sigma_d(R)) \quad \text{cascading selection} \]

\[ R \bowtie (S \bowtie T) \equiv (R \bowtie S) \bowtie T \quad \text{join associativity} \]

\[ \sigma_c(R \bowtie S) \equiv \sigma_c(R) \bowtie S \quad \text{pushing selections} \]

• We can use these equivalences to generate equivalent operator trees
\[ \Pi_{\text{name}, \text{sid}} \left( \sigma_{\text{title}="DB"} \left( \text{Course} \bowtie \left( \text{Students} \bowtie \text{Takes} \right) \right) \right) \]
Algebra v. Calculus

- **Relational Algebra**: More operational; very useful for representing execution plans.
- **Relational Calculus**: More declarative, basis of SQL

- The calculus and algebra have equivalent expressive power (Codd)

A language that can express this core class of queries is called **Relationally Complete**
Relational Algebra & Calculus can’t express all queries

• Tuples in FLIGHTS represent direct flights from departure city to arrival city.
• What about connecting flights?
• A **self-join** is a join of a table with itself
• RA, RC can’t express repeated joins

<table>
<thead>
<tr>
<th>depart</th>
<th>arrive</th>
</tr>
</thead>
<tbody>
<tr>
<td>NYC</td>
<td>Reno</td>
</tr>
<tr>
<td>NYC</td>
<td>Oakland</td>
</tr>
<tr>
<td>Boston</td>
<td>Tampa</td>
</tr>
<tr>
<td>Oakland</td>
<td>Boston</td>
</tr>
<tr>
<td>Tampa</td>
<td>NYC</td>
</tr>
</tbody>
</table>
Next: SQL
SQL Overview

- SQL Preliminaries
- Integrity constraints
- Query capabilities
  - SELECT-FROM-WHERE blocks,
  - Basic features, ordering, duplicates
  - Set ops (union, intersect, except)
  - Aggregation & Grouping
- Nested queries (correlation)
- Null values
- Modifying the database
- Views

Review in the textbook, Ch 5
The SQL Query Language

Structured Query Language

- Developed by IBM (system R) in the 1970s
- Need for a standard since it is used by many vendors
- Evolving standard
  - SQL-86
  - SQL-89 (minor revision)
  - SQL-92 (major revision)
  - SQL-99 (major extensions)
  - SQL-2003 (minor revisions) ...
Two parts of SQL

• Data Definition Language (DDL)
  – Create/alter/delete tables and their attributes
  – establish and modify schema

• Data Manipulation Language (DML)
  – Query and modify database instance
Creating Relations in SQL

- Creates the **Student** relation. Observe that the type (domain) of each field is specified, and enforced by the DBMS whenever tuples are added or modified.

- As another example, the **Takes** table holds information about courses that students take.

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

CREATE TABLE Takes
(sid CHAR(20),
 cid CHAR(20),
 grade CHAR(2))```
Data Types in SQL

• Characters:
  – CHAR(20) -- fixed length
  – VARCHAR(40) -- variable length

• Numbers:
  – BIGINT, INT, SMALLINT, TINYINT
  – REAL, FLOAT -- differ in precision
  – MONEY

• Times and dates:
  – DATE
  – DATETIME

• Others...
Destroying and Altering Relations

**DROP TABLE**  \textbf{Student}

- Destroys the relation \textit{Student}. The schema information \textit{and} the tuples are deleted.

**ALTER TABLE**  \textbf{Student}

\textbf{ADD COLUMN}  \texttt{firstYear} integer

- The schema of \textit{Student} is altered by adding a new field; every tuple in the current instance is extended with a \textit{null} value in the new field.
Integrity Constraints (ICs)

• **IC**: condition that must be true for *any* instance of the database.
  – 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 should only allow legal instances.

• If the DBMS checks ICs, stored data is more faithful to real-world meaning.
  – Avoids data entry errors, too!
Key Constraints

• A set of fields is a \textit{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.
    - If part 2 false: then fields are a \textit{superkey}.
    - If there’s more than one key for a relation, one of the keys is chosen (by DBA) to be the \textit{primary key}.
• E.g., \textit{sid} is a key for Students. (What about \textit{name}?) The set \{\textit{sid, gpa}\} is a superkey.
<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>50000</td>
<td>Dave</td>
<td>dave@cs</td>
<td>19</td>
<td>3.2</td>
</tr>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.3</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@ee</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.7</td>
</tr>
<tr>
<td>53831</td>
<td>Madayan</td>
<td>madayan@music</td>
<td>11</td>
<td>1.8</td>
</tr>
<tr>
<td>53832</td>
<td>Guldu</td>
<td>guldu@music</td>
<td>12</td>
<td>2.0</td>
</tr>
</tbody>
</table>
Specifying Key Constraints in SQL

CREATE TABLE Student
(sid CHAR(20),
 name CHAR(20),
 login CHAR(10),
 age INTEGER,
 gpa REAL,
 UNIQUE (name, age),
 PRIMARY KEY (sid) )

• Possibly many candidate keys (specified using UNIQUE), one of which is chosen as the primary key.
Primary and Candidate Keys in SQL

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

“For a given student and course, there is a single grade.”

CREATE TABLE Takes
(sid CHAR(20),
cid CHAR(20),
grade CHAR(2),
PRIMARY KEY (sid),
UNIQUE (cid, grade) )

“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!
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:
  - Takes(*sid*: string, *cid*: string, *grade*: string)
  - If all foreign key constraints are enforced, **referential integrity** is achieved, i.e., no dangling references.
  - Can you name a data model w/o referential integrity?
    • Links in HTML!
Foreign Keys in SQL

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

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

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>50000</td>
<td>445</td>
<td>A</td>
</tr>
<tr>
<td>53688</td>
<td>483</td>
<td>C</td>
</tr>
<tr>
<td>53666</td>
<td>435</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
</tr>
</thead>
<tbody>
<tr>
<td>50000</td>
<td>Dave</td>
<td>dave@cs</td>
</tr>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@ee</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
</tr>
</tbody>
</table>
Enforcing Referential Integrity

- Consider **Student** and **Takes**; *sid* in **Takes** is a foreign key that references **Student**.
- What should be done if a **Takes** tuple with a non-existent student id is inserted? *(Reject it!)*
- What should be done if a **Student** tuple is deleted?
  - Also delete all Takes tuples that refer to it.
  - Disallow deletion of a Students tuple that is referred to.
  - Set sid in Takes tuples that refer to it to a *default sid*.
  - *(In SQL, also: Set sid in Takes 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)

```sql
CREATE TABLE Takes
(sid CHAR(20),
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.
- 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.