SQL
A Query Language for Relational Databases

John Sieg, UMass Lowell

History
- Structured Query Language, created in 1970s at IBM Research Labs, San Jose
- Has evolved, acquiring more and more features
- SQL 92 is widely supported at various levels.
- SQL:1999 (ANSI and ISO) is current standard.

A Sample Database

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>yob</th>
<th>gpa</th>
<th>sex</th>
<th>major</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>Larry</td>
<td>1985</td>
<td>3.5</td>
<td>M</td>
<td>cs</td>
</tr>
<tr>
<td>s2</td>
<td>Moe</td>
<td>2005</td>
<td>3.2</td>
<td>M</td>
<td>math</td>
</tr>
<tr>
<td>s3</td>
<td>Curly</td>
<td>2004</td>
<td>4</td>
<td>M</td>
<td>cs</td>
</tr>
<tr>
<td>s4</td>
<td>Shemp</td>
<td>2000</td>
<td>4</td>
<td>M</td>
<td>physics</td>
</tr>
</tbody>
</table>

A Sample Database, cont.

CREATE TABLE student

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>yob</th>
<th>gpa</th>
<th>sex</th>
<th>major</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>Larry</td>
<td>1985</td>
<td>3.5</td>
<td>M</td>
<td>cs</td>
</tr>
<tr>
<td>s2</td>
<td>Moe</td>
<td>2005</td>
<td>3.2</td>
<td>M</td>
<td>math</td>
</tr>
<tr>
<td>s3</td>
<td>Curly</td>
<td>2004</td>
<td>4</td>
<td>M</td>
<td>cs</td>
</tr>
<tr>
<td>s4</td>
<td>Shemp</td>
<td>2000</td>
<td>4</td>
<td>M</td>
<td>physics</td>
</tr>
</tbody>
</table>

CREATE TABLE course

<table>
<thead>
<tr>
<th>cid</th>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>Database I</td>
<td>the best course ever</td>
</tr>
<tr>
<td>c2</td>
<td>Database II</td>
<td>the second best course ever</td>
</tr>
<tr>
<td>c3</td>
<td>Compilers</td>
<td></td>
</tr>
<tr>
<td>c4</td>
<td>Poetry</td>
<td></td>
</tr>
</tbody>
</table>

CREATE TABLE enr

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>sec_no</th>
<th>semest</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>c1</td>
<td>201</td>
<td>f2004</td>
<td>a</td>
</tr>
<tr>
<td>s2</td>
<td>c1</td>
<td>201</td>
<td>f2004</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>c2</td>
<td>202</td>
<td>s2005</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>c2</td>
<td>202</td>
<td>s2005</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>c3</td>
<td>201</td>
<td>f2005</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>c4</td>
<td>201</td>
<td>s2005</td>
<td>c</td>
</tr>
<tr>
<td>s2</td>
<td>c1</td>
<td>201</td>
<td>f2005</td>
<td>a</td>
</tr>
<tr>
<td>s3</td>
<td>c2</td>
<td>202</td>
<td>s2005</td>
<td>a</td>
</tr>
</tbody>
</table>

CREATE TABLE, cont’d

CREATE TABLE course_off

<table>
<thead>
<tr>
<th>cid</th>
<th>sec_no</th>
<th>semest</th>
<th>instruct room</th>
<th>time_slot</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>201</td>
<td>f2004</td>
<td>Olsen 410</td>
<td>T 5-7:10</td>
</tr>
<tr>
<td>c1</td>
<td>201</td>
<td>f2004</td>
<td>Olsen 410</td>
<td>T 5-7:30</td>
</tr>
<tr>
<td>c2</td>
<td>202</td>
<td>s2005</td>
<td>Ball 210</td>
<td>M 5-7:30</td>
</tr>
<tr>
<td>c2</td>
<td>202</td>
<td>s2005</td>
<td>Ball 212</td>
<td>M 5-7:30</td>
</tr>
<tr>
<td>c3</td>
<td>201</td>
<td>f2005</td>
<td>Olsen 115</td>
<td>M 3-4:30</td>
</tr>
<tr>
<td>c4</td>
<td>201</td>
<td>s2005</td>
<td>Olsen 115</td>
<td>M 3-4:30</td>
</tr>
</tbody>
</table>

CREATE TABLE enr

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>sec_no</th>
<th>semest</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>c1</td>
<td>201</td>
<td>f2004</td>
<td>a</td>
</tr>
<tr>
<td>s2</td>
<td>c1</td>
<td>201</td>
<td>f2004</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>c2</td>
<td>202</td>
<td>s2005</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>c2</td>
<td>202</td>
<td>s2005</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>c3</td>
<td>201</td>
<td>f2005</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>c4</td>
<td>201</td>
<td>s2005</td>
<td>c</td>
</tr>
<tr>
<td>s2</td>
<td>c1</td>
<td>201</td>
<td>f2005</td>
<td>a</td>
</tr>
<tr>
<td>s3</td>
<td>c2</td>
<td>202</td>
<td>s2005</td>
<td>a</td>
</tr>
</tbody>
</table>
Domains

CREATE DOMAIN SID_DOMAIN CHAR(5);
CREATE DOMAIN SNAME_DOMAIN CHAR(16);
...
CREATE TABLE student
  (sid      SID_DOMAIN,
   sname    SNAME_DOMAIN,
   ...)

Domain Constraints

CREATE DOMAIN SEX_DOMAIN CHAR(1)
CONSTRAINT CHECK_SEX
CHECK
  ( VALUE IN ('m', 'f'))

DROP TABLE

DROP TABLE student

removes the student table, including schema

Indices (not part of SQL standards)

CREATE CLUSTERED INDEX StudentInd
ON student (sid)
builds an index on student.sid

DROP student.StudentInd
deletes the index

Updates

INSERT INTO student (sid, sname, yog, gpa, sex, major)
VALUES
  ('s1', 'Larry', 'grad', 4.0, 'm', 'cs');

UPDATE student
SET major = 'cs'
WHERE student.gpa = 4.0;

DELETE FROM student
WHERE yog = '2005';

Syntax - The SELECT Statement

SELECT column_list
FROM tablename_list
[ WHERE search_condition ]
[ GROUP BY column_list ]
[ HAVING group_condition ]
[ ORDER BY column_list ]
SQL Select Statement

Give the names of Computer Science (CS) majors.

```
SELECT sname
FROM student
WHERE major = 'cs'
```

SQL Select Statement, cont.

Give the names of students who have received at least one grade of ‘A’.

```
SELECT sname
FROM student
WHERE enr.grade = 'a'
AND student.sid = enr.sid
```

SQL Select Statement, cont.

Give the names of students who have received at least one grade of ‘A’.

```
SELECT DISTINCT sname
FROM student, enr
WHERE enr.grade = 'a'
AND student.sid = enr.sid
```

SQL UNIONs

Give the CIDs of courses that either have prerequisites or are prerequisites for other courses or both.

```
SELECT cid FROM prereq
UNION
SELECT prereq_cid FROM prereq
```

- Duplicates not returned. UNION ALL returns duplicates.
- SQL also has INTERSECT and EXCEPT operators.
- Union compatible => same number and types of columns

Table Aliases

Give the names of courses that have prerequisites and the names of the prerequisites.

```
SELECT c1.cname AS courseName,
   c2.cname AS prereq
FROM course c1, prereq, course c2
WHERE c1.cid = prereq.cid
AND prereq.prereq_cid = c2.cid
```

SQL Views

Create a view that joins together students with their enrollments.

```
CREATE VIEW student_enr
AS
SELECT sid, sname, major, yog, cid, sec_no, semester, grade
FROM student, enr
WHERE student.sid = enr.sid
```
Querying a View

Give the names of students who have received at least one grade of ‘A’.

```
SELECT sname
FROM student_enr
WHERE grade = 'a'
```
**SQL Select Statement, cont.**

Give the names of students who have received at least one grade of ‘A’.

```sql
SELECT name
FROM student
WHERE EXISTS
  (SELECT *
   FROM enr
   WHERE student.sid = enr.sid
   AND enr.grade = 'a')
```

**GROUP BY**

Give each (enrolled) student's sid and number of courses he has been enrolled in.

```sql
SELECT sid, COUNT(DISTINCT cid) AS num
FROM enr
GROUP BY sid
```

**HAVING**

Give the SIDs of students who have been enrolled in more than two different courses.

```sql
SELECT sid
FROM enr
GROUP BY sid
HAVING COUNT(DISTINCT cid) > 2
```

**SQL Select Statement, cont.**

Give the SIDs of students who have received A's in all their courses.

```sql
SELECT sid
FROM student
WHERE NOT EXISTS
  (SELECT *
   FROM enr
   WHERE enr.sid = student.sid
   AND enr.grade <> 'a')
```

**SQL Select Statement, cont.**

Give the names of students who have enrolled in every course.

```sql
SELECT name
FROM student
WHERE NOT EXISTS
  (SELECT *
   FROM course
   WHERE NOT EXISTS
     (SELECT *
      FROM enr
      WHERE student.sid = enr.sid
      AND enr.cid = course.cid))
```

**SQL Select Statement, cont.**

Give the names of students who have received at least one grade of ‘A’.

```sql
SELECT name
FROM student JOIN enr
ON student.sid = enr.sid
WHERE enr.grade = 'a'
```
Outer Joins
Tell me everything about students and their enrollments. Include students who don't have any enrollments.

```sql
SELECT *
FROM student LEFT OUTER JOIN enr
ON student.sid = enr.sid
```

A Left Outer Join

<table>
<thead>
<tr>
<th>st</th>
<th>sid</th>
<th>name</th>
<th>major</th>
<th>gpa</th>
<th>sec</th>
<th>sem</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Larry</td>
<td>grad</td>
<td>3.5</td>
<td>M</td>
<td>cs</td>
<td>s1</td>
<td>c1</td>
</tr>
<tr>
<td>2</td>
<td>Mike</td>
<td>2003</td>
<td>3.2</td>
<td>M</td>
<td>math</td>
<td>s2</td>
<td>c1</td>
</tr>
<tr>
<td>3</td>
<td>Mike</td>
<td>2003</td>
<td>3.2</td>
<td>M</td>
<td>math</td>
<td>s2</td>
<td>c2</td>
</tr>
<tr>
<td>4</td>
<td>Larry</td>
<td>2003</td>
<td>3.5</td>
<td>M</td>
<td>CS</td>
<td>s1</td>
<td>c2</td>
</tr>
<tr>
<td>5</td>
<td>Curly</td>
<td>2004</td>
<td>4.0</td>
<td>M</td>
<td>math</td>
<td>s3</td>
<td>c3</td>
</tr>
<tr>
<td>6</td>
<td>Curly</td>
<td>2004</td>
<td>4.0</td>
<td>M</td>
<td>math</td>
<td>s3</td>
<td>c4</td>
</tr>
</tbody>
</table>

Cartesian Products
Show me all pairings of students with enrollments, whether related or not.

```sql
SELECT name
FROM student CROSS JOIN enr
```

Triggers / Active Databases

CREATE TRIGGER RejectPoorStudents
AFTER INSERT ON Student /* Event */
WHEN Student.salary=0 /* Cond'n */
FOR EACH ROW
INSERT INTO RejectLog /* Action */
VALUES (sid, name, sex)

Embedded SQL: Declaring a Cursor

```sql
DECLARE student_cursor CURSOR
FOR
SELECT S.name, S.yog
FROM student S
WHERE S.gpa > :ext_gpa;
```

Embedded SQL: Retrieving from a Cursor

```sql
OPEN student_cursor
...
FETCH student_cursor
    INTO :ext_name, :ext_yog
...
CLOSE student_cursor
```
**Dynamic SQL**

```java
char user_query[];
// Build an SQL query
// dynamically.
getquery(user_query);
EXEC SQL PREPARE SqlStatement
  FROM :user_query;
EXEC SQL EXECUTE SqlStatement
```

**Translating a Program with Embedded SQL**

Program with embedded SQL

Preprocessor

C/C++/… with function calls

Compiler

Executable (linked with run-time code of DBMS)

**SQLJ Example**

```java
String Sid;
// Missing code: Get Sid from user.
#sql iterator StudIteratorClass (name, gpa);
StudIteratorClass StudentIter;
#sql StudentIter =
  {SELECT name, gpa
   FROM student
   WHERE sid = :Sid};
```

**SQLJ Example, cont’d**

```java
while (StudentIter.next())
{
    System.out.println(
        StudentIter.sname() + "  "
        + StudentIter.gpa());
}
StudentIter.close();
```

**Competing Standard APIs: ODBC and JDBC**

- Function calls are standardized, unlike embedded SQL.
- Access to different data sources is mediated by ODBC/JDBC drivers.
- Unlike with embedded SQL or SQLJ, binding to database system is done at run-time.

**Conclusions**

- SQL 92 is an ugly language. Alternatives such as Quel are cleaner.
- SQL:1999 cleans up some of the orthogonality issues and adds many new features. We’ll see some of the new features of SQL:1999 later.
- SQL will continue to grow. Current work includes extensions to query XML data.