CS2Bh: Current Technologies

Introduction to XML and Relational Databases

Spring 2005

Introduction to SQL

SQL

The most used “programming language” – extracting data

- ✓ Data Definition Language (DDL)
  - Create/delete/modify relations (and views)
  - Define integrity constraints (ICs)
  - Grant/revoke privileges (security)
- ✓ Data Manipulation Language (DML)
  - Update language
    - Insert/delete/modify tuples
    - Interact with IC’s
  - Query language
    - Relationally complete!
    - Beyond relational algebra!

SQL: DDL

Create relations:

CREATE TABLE Students (sid INTEGER, sname CHAR(10), gpa REAL)

CREATE TABLE Courses (cid INTEGER, cname CHAR(10), credit INTEGER, instructor CHAR(10))

CREATE TABLE Enroll (sid INTEGER, cid INTEGER, grade CHAR(1))

Domain types: INTEGER, INTEGER(n, d), CHAR, DATE, …

DATE: usually specified in string format, e.g., '12-FEB-2000'

DDL (cont'd)

Deleting relations (and the tuples as well):

DROP TABLE Students

DROP TABLE Students CASCADE CONSTRAINTS

Drops all referential integrity constraints that refer to primary keys in the dropped table.

Question: how to specify integrity constraints in a schema?

IC: keys

Key for a relation: a *minimum* set of fields that uniquely identify a tuple

- ✓ Candidate keys: possibly many, specified using UNIQUE
- ✓ Primary key: unique, specified using PRIMARY KEY

```
CREATE TABLE Students
(sid INTEGER,
 sname CHAR(10),
 gpa REAL,
 PRIMARY KEY (sid))
```

```
CREATE TABLE Students
(sid INTEGER,     -- DEFAULT 001
 sname CHAR(10),
 gpa REAL,
 PRIMARY KEY (sid),
 UNIQUE (sname))
```

Exercise:

- ✓Specify cid as the primary key for Courses.
- ✓Specify sid, cid as the primary key for Enroll.

IC: foreign keys

Foreign keys: a set of fields in one relation R that is used to 'refer' to another relation S

- ✓ Fields should be a key (primary key) for S
- ✓ In tuples of R, field values must match values in some S tuple – no dangling pointers

```
CREATE TABLE Enroll
    (sid INTEGER,
     cid INTEGER,
    grade CHAR(1),
    PRIMARY KEY (sid, cid),
    FOREIGN KEY (sid) REFERENCES Students,
    FOREIGN KEY (cid) REFERENCES Courses)
```

IC: other constraints

- *check condition*, e.g., gpa INTEGER(1, 2) check (gpa < 4.0)
- *not null,* e.g., sname CHAR(20) not null

Null values

- Attribute values in a tuple are sometimes *unknown* or *inapplicable* (e.g., no spouse's name for a single). These are treated as a special value: null
- Keys cannot have null values (but foreign keys can)
- Three-valued logic:
  - Comparison operations (e.g., >)
    e.g., 3 < null -- *unknown*
  - Logic connectives (e.g., AND, OR, NOT)
    - false AND unknown? False
    - true OR unknown? True
    - false OR unknown? unknown

Enforcing referential integrity

Recall deletion/update strategies: to delete a Students tuple,

- ✓ Also delete all Enroll tuples that refer to it (CASCADE).
- ✓ Rejection (NO ACTION).
- ✓ Set sid in Enroll tuples that refer to it to a default sid (null): (SET NULL/SET DEFAULT).

SQL supports all of these. Default is NO ACTION: rejection.

```
CREATE  TABLE   Enroll
(sid  INTEGER,
 cid  INTEGER,
 grade CHAR(1),
 PRIMARY KEY (sid, cid),
 FOREIGN KEY  (sid)  REFERENCES  Students,
 FOREIGN KEY  (cid)  REFERENCES  Courses
        ON DELETE CASCADE
        ON UPDATE  SET DEFAULT)
```


Update language -- inserting new tuples

Single tuple insertion:

```
INSERT INTO Students   (sid, sname, gpa)
VALUES (001,  'John',  3.6)
INSERT INTO Students   (sid, sname, gpa)
VALUES (002,  'Mary',  2.6)
INSERT INTO Students   (sid, sname, gpa)
VALUES (003,  'Grace',  4.0)
```

An insert command that causes an IC violation is rejected!

Question: what if we tried to insert (003, 'Joe', 4.0)?

Other operations: multiple record insertion, deletion, modification– we'll come back to this topic.

Simple SQL queries

1. **Projection**. Find the names of Students.

   Recall π_{sname} (Students)

   ```
   SELECT  sname
   FROM    Students
   ```

2. **Selection**. Find the courses taught by Fan.

   Recall $\sigma_{\text{instructor = 'fan'}}$ (Courses)

   ```
   SELECT  *
   FROM    Courses
   WHERE   instructor = 'fan'
   ```


Project/Select

Find the names of students with gpa > 3.0.

$\pi_{\text{sname}}(\sigma_{\text{gpa > 3.0}}\text{Students})$

SQL does not eliminate duplicates unless you ask explicitly!

```
SELECT  sname
FROM    Students
WHERE   gpa  > 3.0
```

```
SELECT  DISTINCT  sname
FROM    Students
WHERE   gpa  > 3.0
```

sid	sname	gpa
001	john	3.6
002	mary	2.6
003	grace	4.0
004	john	3.2

sname
john
grace
john

sname
john
grace

Basic syntax of SQL queries

SELECT **[DISTINCT]** *attribute-list*
FROM *relation-list*
WHERE *condition*

- *relation-list* is a list of relation names, possibly with an range variable after some name.
- *attribute-list* is a list of attributes of relations in *relation-list*. A **'*'** can be used to denote 'all attributes'. You may rename the attributes.
- condition
  - comparison: *Attr op const* or *Attr op Attr.*
  - Op: <, >, =, <=, >=, <>.
  - boolean connectives: AND, OR, NOT.

Basic Syntax of SQL (cont'd)

Other conditions: like (Oracle) performs pattern matching in string data, e.g., sname like 'f%' (%: one or more characters, _: one character)

- DISTINCT is an optional keyword indicating that the answer should not contain duplicates. Default is that duplicates are *not* eliminated.

Conceptual evaluation strategy

SELECT [DISTINCT] *attribute-list*
FROM *relation-list*
WHERE *condition*

- ✓ Compute the cross-product of *relation-list*
- ✓ Discard resulting tuples if they do not satisfy *condition*
- ✓ Delete attributes that are not in *attribute-list*
- ✓ If DISTINCT is present, eliminate duplicate tuples.

This strategy is probably the least efficient way to compute a query! An optimizer will find more efficient strategies to compute the *same answers.*

Example of conceptual evaluation -- product

SELECT *
FROM Students, Enroll

sid	sname	gpa
001	john	3.6
002	mary	2.6
003	grace	4.0

sid	cid	grade
003	166	A
003	CS2	A

Students.sid	sname	gpa	sid	cid	grade
001	john	3.6	003	166	A
002	mary	2.6	003	166	A
003	grace	4.0	003	166	A
001	john	3.6	003	CS2	A
002	mary	2.6	003	CS2	A
003	grace	4.0	003	CS2	A

Example of conceptual evaluation – join

Find the names of Students who are taking CS2.

$\pi_{sname}(\sigma_{cid = CS2}(Students \bowtie Enroll))$

```
SELECT   sname
FROM     Students,  Enroll
WHERE    Students.sid = Enroll.sid   AND  Enroll.cid = CS2
```

Questions: what is the result?

sid	sname	gpa
001	john	3.6
002	mary	2.6
003	grace	4.0
004	john	CS2

sid	cid	grade
003	166	A
003	CS2	A
001	CS2	B
004	CS2	C

A note on range variables

Find the names of Students who are taking CS2.

It is a bit awkward to write Students.sid

```
SELECT   sname
FROM     Students,  Enroll
WHERE    Students.sid = Enroll.sid   AND  Enroll.cid = CS2
```

We can rewrite it using range variables:

```
SELECT   S.sname
FROM     Students S,  Enroll E
WHERE    S.sid = E.sid   AND  E.cid = CS2
```

- ✓ Really needed only if the same relation appears twice in the FROM clause.
- ✓ It is good style, however, to use range variables all the time!

Example

find the names of students who do not have the highest GPA.

```
SELECT   S1.sname
FROM     Students S1,  Students S2
WHERE    S1.gpa < S2.gpa
```


More on joins

There is no explicit natural join in SQL.

Find the names of Students who are taking a course taught by Fan.

π_{sname} (Students $\bowtie$ Enroll $\bowtie$ $\sigma_{\text{instructor = 'fan'}}$ (Courses))

```
SELECT   S.sname
FROM     Students S,  Enroll E,  Courses  C
WHERE    S.sid = E.sid   AND  E.cid = C.cid  AND
         C.instructor = 'fan'
```

- ✓ SQL supports conditional join. Recall that natural join is a special case of conditional join.
- ✓ To do natural join, you have to explicitly list all the equality conditions, i.e., equality on *all* the common fields.

exercise

1. Find the instructors of the courses being taken by Grace.
2. Find the names of students who are taking at least one course.
3. Find the names of students who are taking at least two courses.

Union

Find the names of Students who are taking a course taught by Poe or Fan.

```
SELECT   S.sname
FROM     Students S,  Enroll E,  Courses  C
WHERE    S.sid = E.sid   AND  E.cid = C.cid
         AND  C.instructor = 'fan'
UNION                    /* UNION ALL reserves duplicates
SELECT   S.sname
FROM           Students S,  Enroll E,  Courses  C
WHERE    S.sid = E.sid   AND  E.cid = C.cid  AND
         C.instructor = 'poe'
```

Union and OR

Of course, you may write this as

```
SELECT     S.sname
FROM    Students S,  Enroll E,  Courses  C
WHERE      S.sid = E.sid   AND  E.cid = C.cid  AND
                    (C.instructor = 'fan'  OR C.instructor = 'poe')
```


What does 'union compatible' mean?

```
SELECT   S.sid
FROM      Students S,  Enroll E,  Courses  C
WHERE   S.sid = E.sid   AND  E.cid = C.cid
        AND  C.instructor = 'fan'
UNION
SELECT   S.sname
FROM      Students S,  Enroll E,  Courses  C
 WHERE    S.sid = E.sid   AND  E.cid = C.cid
         AND  C.instructor = 'poe'
```

What is the result of this query?

By SQL standard, this is an error.

Intersection

Find the ids of Students who are taking a course taught by Poe and a course taught by Fan.

π_{sid} (Enroll $\bowtie$ $\sigma_{instructor = 'poe'}$ (Courses)) $\cap$

π_{sid} (Enroll $\bowtie$ $\sigma_{instructor = 'fan'}$ (Courses))

```
SELECT    S.sid
FROM          Enroll E,  Courses  C
WHERE     E.cid = C.cid  AND  C.instructor = 'fan'
INTERSECT
SELECT    S.sid
FROM          Enroll E,  Courses  C
WHERE     E.cid = C.cid  AND  C.instructor = 'poe'
```


question

Find the names of Students who are taking a course taught by Poe and a course taught by Fan. -- Nested queries!

Set difference

Find the ids of Students who are not taking CS2.

π_{sid} (Students) $-$ π_{sid} ($\sigma_{cid = 'CS2'}$ (Enroll)

```
SELECT   sid
FROM     Students
DIFFERENCE                    /*  EXCEPT
SELECT   E.sid
FROM     Enroll E
 WHERE   E.cid = 'CS2'
```

Some systems use MINUS (or EXCEPT) instead of DIFFERENCE, such as Oracle

Question: how to do set difference/intersection if your system does not support them? Nested queries!

Nested queries -- Intersection

Find the names of Students who are taking a course taught by Poe and a course taught by Fan.

```
SELECT     S.sname
FROM       Students S,  Enroll E,  Courses  C
WHERE   S.sid = E.sid   AND  E.cid = C.cid  AND
           C.instructor = 'fan' AND
           S.sid  IN  (SELECT    S2.sid
                       FROM    Students S2,  Enroll E2,  Courses  C2
                      WHERE   S2.sid = E2.sid   AND  E2.cid =
                                 C2.cid  AND  C2.instructor = 'poe')
```

A very powerful feature of SQL: a WHERE clause can itself contain a SQL query! In fact, so can FROM and HAVING clause.

The query in WHERE clause is called a *subquery*.

Nested queries -- Set difference

Find the names of Students who are not taking CS2.

```
SELECT   S.sname
FROM     Students  S
WHERE    S.sid  NOT IN  (SELECT   E.sid
                        FROM     Enroll E
                        WHERE    E.cid = 'CS2')
```

We have learned

- ✓ IN, NOT IN
- ✓ What else?
  - EXISTS, NOT EXISTS
  - EXISTS UNIQUE, NOT EXISTS UNIQUE
  - Set comparison, e.g., > ANY, > ALL

Nested queries with correlation

EXISTS, NOT EXISTS

Find the names of students who are taking a course taught by Poe and a course taught by Fan.

```
SELECT   S.sname
FROM     Students  S, Enroll  E,  Courses C
WHERE    S.sid = E.sid  AND  E.cid = C.cid
         AND C.instructor = 'fan' AND
         EXISTS   (SELECT   S2.sname
                   FROM     Students S2,  Enroll  E2,  Courses C2
                   WHERE    S2.sid = E2.sid  AND   E2.cid = C2.cid
                   AND      C2.instructor = 'poe' AND  S2.sid = S.sid)
```

Correlation: note *S2.sid = S.sid.*

Nested queries with correlation

```
SELECT   S.sname
FROM     Students  S, Enroll  E,  Courses C
WHERE    S.sid = E.sid  AND  E.cid = C.cid
         AND C.instructor = 'fan' AND
         EXISTS  (SELECT   S2.sname
                  FROM     Students S2,  Enroll  E2,  Courses C2
                  WHERE    S2.sid = E2.sid  AND   E2.cid = C2.cid
                  AND      C2.instructor = 'poe' AND  S2.sid = S.sid)
```

In general, subquery must be re-computed for each Students tuple S -- nested loop.

- ✓ NOT EXISTS: empty set testing.

Exercise: Find the names of Students who are not taking CS2.

More on nested queries

EXISTS UNIQUE, NOT EXISTS UNIQUE: checking for duplicate tuples. True only if no two tuples appear more than once in the answer to the subquery.

Find the names of students who either do not take CS2, or don't take 166.

```
SELECT   S.sname
FROM     Students  S
WHERE    EXISTS  UNIQUE (SELECT    DISTINCT   E.sid
                          FROM      Enroll  E
                          WHERE     S.sid = E.sid   AND  E.cid = CS2
                          UNION  ALL
                          SELECT    DISTINCT  E.sid
                          FROM      Enroll  E
                          WHERE     S.sid = E.sid  AND  E.cid = 166)
```

Set comparison operations

op ANY, op ALL, where op: >, <, =, <>, >=, <=.

ANY: there exists some (existential). ALL: for all (every), universal.

Find the names of students whose GPAs are higher than that of some student called Joe.

```
SELECT   S.sname
FROM     Students  S
WHERE    S.gpa  > ANY (SELECT   S2.gpa
                         FROM     Students  S2
                         WHERE    S2.sname = 'Joe')
```


Set comparison -- universal

Question: Find the names of students whose GPAs are higher than that of every student called Joe.

What if there is no student called Joe?

- ✓ *S.gpa* > *ANY* (…) returns false
- ✓ *S.gpa* > *ALL* (…) returns true.

Division (universal quatification)

Find the sids of students who are taking all courses.

Given a student tuple S, compute the set of cids of the courses that S is not taking.

```
SELECT   C.cid
FROM     Courses  C
EXCEPT
SELECT   E.cid
FROM      Enroll  E
WHERE     S.sid = E.sid
```


Division (cont'd)

S is put in the answer if and only if the set is empty!

```
SELECT   S.sname
FROM     Students  S
WHERE    NOT EXISTS (SELECT   C.cid
                     FROM     Courses  C
                     EXCEPT
                     SELECT   E.cid
                     FROM      Enroll  E
                     WHERE     S.sid = E.sid)
```

More on division

Find the names of students who are taking all courses taught by Fan.

```
SELECT   S.sname
FROM     Students  S
WHERE    NOT EXISTS (SELECT   C.cid
         FROM     Courses  C
         WHERE    C.instructor = 'fan'
EXCEPT
SELECT   E.cid
         FROM     Enroll  E
         WHERE    S.sid = E.sid)
```

Exercise: rewrite the query without using DIFFERENCE.

Using expressions as relation names

Find the names of students who are taking CS2.

```
SELECT   S.sname
FROM     Students  S,  (SELECT   E.sid
                       FROM     Enroll E
                       WHERE    E.cid = CS2)   AS   temp
WHERE    S.sid = temp.sid
```

Naming temporary (intermediate) relation: FROM clause can also contain subquery

Using expressions as relation names (cont'd)

How to rename attributes?

```
SELECT   S.sname  AS name      //Or  name = S.sname
FROM     Students  S,  (SELECT   E.sid
                        FROM     Enroll E
                        WHERE    E.cid = CS2)   AS   temp
WHERE    S.sid = temp.sid
```


Aggregate functions – non-algebraic operators

Significant extension of relational algebra:

- ✓ COUNT(*), COUNT([DISTINCT] (A)),
- ✓ SUM([DISTINCT] (A)),
- ✓ AVG ([DISTINCT](A)),
- ✓ MAX(A), MIN(A).

Here A is an attribute.

Examples:

```
SELECT COUNT(*)
FROM   Students
```

```
SELECT AVG(S.gpa)
FROM   Students S
```

```
SELECT MAX(S.gpa)
FROM   Students  S
```

Aggregate functions – non-algebraic operators

COUNT(*), COUNT([DISTINCT] (A)),SUM([DISTINCT] (A)), AVG ([DISTINCT](A)), MAX(A), MIN(A).

Examples:

```
SELECT COUNT(DISTINCT (S.sname))
FROM    Students S, Enroll E
WHERE  S.cid = E.sid AND E.cid = CS2
```

```
SELECT S.sname
FROM    Students S
WHERE  S.gpa = (SELECT MAX(S2.gpa)
                FROM Students S2)
```


Aggregate functions in SELECT clause – Introduction to GROUP BY

Find the name and GPA of the student(s) with highest GPAs.

```
SELECT S.sname, MAX(S.gpa)
FROM    Students S
```

```
SELECT S.sname, S.gpa
FROM    Students S
WHERE  S.gpa = (SELECT MAX(S2.gpa)
                FROM Students S2)
```

The first query is illegal: If SELECT clause uses an aggregate function, it must contain only aggregate operation unless the query contains GROUP BY clause.

Aggregate functions in SELECT clause – Introduction to GROUP BY

Why GROUP BY? Sometimes we want to apply aggregate functions to each of several groups.

Example: 'Find the number of students taking CS2 for each grade.'

For G in [A, A-, B, B-, C, C-, D, I, F], we have to write a query that looks like:

```
SELECT   COUNT(E.sid)
FROM     Enroll E
WHERE    E.cid = CS2  AND  E.grade = 'A'
```

But in general, we don't know how many values (groups) we may have!

Group by

For each grade, find the number of CS2 students receiving that grade.

```
SELECT    E.grade, COUNT(E.sid)
FROM      Enroll E
WHERE     E.cid = CS2
GROUP BY  E.grade
```

For each grade higher than 'F', find the number of CS2students receiving that grade.

```
SELECT    E.grade, COUNT(E.sid)
FROM      Enroll E
WHERE     E.cid = CS2
GROUP BY  E.grade
Having    E.grade > 'F'
```

Queries with GROUP BY

SELECT [**DISTINCT**] *target-list*
FROM *relation-list*
WHERE *condition*
GROUP BY *grouping-list*
HAVING *group-qualifications*

- *target-list* contains
  - *attribute lists*
  - terms with aggregate functions (e.g., MAX(S.gpa))
- *grouping-list* is a list of attributes used to determine groups. Attributes in *attribute-list* MUST be also in *grouping-list.* E.g., E.grade.

  A group is a set of tuples that have the same values for all attributes in *grouping-list.*
- *group-qualifications* restrict what groups we want. It is optional.

Queries with GROUP BY

SELECT [**DISTINCT**] *target-list*
FROM *relation-list*
WHERE *condition*
GROUP BY *grouping-list*
HAVING *group-qualifications*

- *group-qualifications* restrict what groups we want. It is optional. An attribute appears in *group-qualifications* MUST be also in *grouping-list!*

Conceptual evaluation

SELECT **[DISTINCT]** *target-list*
FROM *relation-list*
WHERE *condition*
GROUP BY *grouping-list*
HAVING *group-qualifications*

- ✓ The cross-product of *relation-list* is computed, tuples that fail *condition* are discarded, attributes that are not in *attribute-list* are threw away, and the remaining tuples are partitioned into groups by the value of attributes of *grouping-list.*
- ✓ *group-qualifications* are then applied to eliminate some groups.
- ✓ One answer per group is generated per qualifying group.

example

Find the number of students taking CS2for each grade.

```
SELECT    E.grade, COUNT(E.sid)
FROM      Enroll E
WHERE  E.cid = CS2
GROUP BY  E.grade
```

Is the following correct?

```
SELECT    E.grade, COUNT(E.sid)
FROM      Enroll E
GROUP BY  E.grade
HAVING    E.cid = CS2
```

More on GROUP BY

✓ Is the following correct?

```
SELECT   E.sid, E.grade, COUNT(E.sid)
FROM     Enroll E
WHERE    E.cid = CS2
GROUP BY E.grade
```

✓ Find the average GPA of students for each course with credit > 2 taught by each instructor.

```
SELECT   C.instructor, C.credit, AVG(S.gpa)
FROM     Students S, Enroll E, Courses C
WHERE    S.sid = E.sid AND E.cid = C.cid
GROUP BY C.instructor, C.credit
HAVING   C.credit > 2
```


ORDER BY

Find the names and grades of CS2students, ordered by their grades.

```
SELECT S.sname, E.grade
FROM   Students S, Enroll E
WHERE  S.sid = E.sid AND E.cid = 'CS2'
ORDERED BY E.grade
```

Find the names and grades of CS2students, ordered first by their grades and within each grade, ordered by names.

```
SELECT S.sname, E.grade
FROM   Students S, Enroll E
WHERE  S.sid = E.sid AND E.cid = 'CS2'
ORDERED BY E.grade, S.sname
```

✓ ASC and DESC, e.g., E.grade ASC, S.sname DESC

✓ The default is ASC

Summary – SQL query language

- Basic queries: relational completeness.
  - IN, NOT IN
  - EXISTS, NOT EXISTS, EXISTS UNIQUE, NOT EXISTS UNIQUE
- Beyond relational algebra
  - Aggregate functions
  - GROUP BY and HAVING
- What else?
  - Embedded SQL (e.g., Java, C++, …)
  - Triggers and active databases.

More on SQL update language

- Multiple record insertion: Let's define a new relation Persons

```
CREATE TABLE Persons (id INTEGER,  name CHAR(10))
INSERT INTO Persons (id, name)
SELECT S.sid, S.sname
FROM    Students S
WHERE  S.gpa > 1
```

- Deletion

```
DELETE
FROM    Students S
WHERE  S.gpa < 1
```

More on SQL update language

- Modification

```
UPDATE  Students S
SET     S.gpa = S.gpa + 1;
WHERE   S.gpa < 1
```


More on SQL DDL -- views

- A view is just a relation, but we store a *definition*, instead of a set of tuples

```
CREATE VIEW ClassCS2(sname, grade)
            AS SELECT S.sname, E.grade
               FROM    Students S, Enroll E
               WHERE   S.sid = E.sid  AND  E.cid = CS2
```

- Views can be dropped

```
DROP VIEW  classCS2
```

- Views can be used to present necessary information, while hiding details in underlying the relation(s). Security considerations.

More on SQL DDL -- views

- Queries on views: query modification

```
SELECT S.grade,  COUNT(DISTINCT(S.name))
FROM    ClassCS2S
GROUP BY   S.grade
```

- Updates on views

Summary

- Query language: basic queries, nested queries, aggregate functions, group by
- SQL DDL: schema, key, foreign keys, update/delete policies, views.

  What else? General ICs, security.
- SQL update language: insert, delete, update
- **SKILL: 'Programming' in SQL!**

what we have learned

An introduction to XML and XML querying

- XML basics
- Document Type Definition (DTD)
- XPath
- XQuery and XSLT

An introduction to databases

- Relational data model
- Relational algebra
- SQL