Announcements (February 1)

- Reading assignment for this week
  - System R paper and Lomet’s B+-tree tricks
  - Due Thursday night
- Homework #1 due in 7 days
- No class next Thursday (February 10)
  - Instructor out of town for a program committee meeting

Summary of SQL features covered so far

- Basic modeling features
  - Bags, NULL’s
- Schema features
  - CREATE/DROP TABLE
- Query features
  - SELECT-FROM-WHERE statements, set and bag operations, table expressions, aggregation and grouping
  - Next: subqueries

Scalar subqueries

- A query that returns a single row can be used as a value in WHERE, SELECT, etc.
- Example: students at the same age as Bart
  ```sql
  SELECT * FROM Student
  WHERE age = (SELECT age FROM Student WHERE name = 'Bart');
  ```
  What’s Bart’s age?
  - Runtime error if subquery returns more than one row
  - Under what condition will this runtime error never occur?
    - name is a key of Student
  - What if subquery returns no rows?
    - Return NULL

IN subqueries

- \( x \) IN (subquery) checks if \( x \) is in the result of subquery
- Example: students at the same age as (some) Bart
  ```sql
  SELECT * FROM Student
  WHERE age IN (SELECT age FROM Student WHERE name = 'Bart');
  ```

EXISTS subqueries

- EXISTS (subquery) checks if the result of subquery is non-empty
- Example: students at the same age as (some) Bart
  ```sql
  SELECT *
  FROM Student AS s
  WHERE EXISTS (SELECT * FROM Student
  WHERE EXISTS (SELECT * FROM Student
  WHERE age = s.age
  AND name = 'Bart');
  ```
  - It is a correlated subquery—a subquery that references tuple variables in surrounding queries

SQL: Part II

CPS 216
Advanced Database Systems
Operational semantics of subqueries

- `SELECT * FROM Student AS s WHERE EXISTS (SELECT * FROM Student WHERE name = 'Bart' AND age = s.age);`

- For each row `s` in `Student`
  - Evaluate the subquery with the appropriate value of `s.age`
  - If the result of the subquery is not empty, output `s.*`

- The DBMS query optimizer may choose to process the query in an equivalent, but more efficient way (example?)

Scoping rule of subqueries

- To find out which table a column belongs to
  - Start with the immediately surrounding query
  - If not found, look in the one surrounding that; repeat if necessary
- Use `table_name.column_name` notation and `AS` (renaming) to avoid confusion

Another example

```
SELECT * FROM Student AS s
WHERE EXISTS
  (SELECT * FROM Enroll e
   WHERE SID = s.SID
   AND EXISTS
     (SELECT * FROM Enroll
      WHERE SID = s.SID
      AND CID <> e.CID));
```

Students who are taking at least two courses

Quantified subqueries

- A quantified subquery can be used as a value in a `WHERE` condition
- Universal quantification (for all):
  ... `WHERE x op ALL(subquery)` ...
  - True iff for all `t` in the result of `subquery`, `x op t`
- Existential quantification (exists):
  ... `WHERE x op ANY(subquery)` ...
  - True iff there exists some `t` in the result of `subquery` such that `x op t`

Beware
- In common parlance, “any” and “all” seem to be synonyms
- In SQL, `ANY` really means “some”

Examples of quantified subqueries

- Which students have the highest GPA?
  - `SELECT * FROM Student WHERE GPA >= ALL (SELECT GPA FROM Student);`
  - `SELECT * FROM Student WHERE NOT EXISTS (SELECT * FROM Student WHERE GPA > s.GPA);`
  - `SELECT * FROM Student WHERE SID NOT IN (SELECT s1.SID FROM Student AS s1, Student AS s2 WHERE s1.GPA < s2.GPA);`

- Use `NOT` to negate a condition

More ways of getting the highest GPA

- Which students have the highest GPA?
  - `SELECT * FROM Student AS s WHERE NOT EXISTS (SELECT * FROM Student WHERE GPA > s.GPA);`
  - `SELECT * FROM Student WHERE SID NOT IN (SELECT s1.SID FROM Student AS s1, Student AS s2 WHERE s1.GPA < s2.GPA);`
Summary of SQL features covered so far

- Basic modeling features
  - Bags, NULL's
- Schema features
  - CREATE/DROP TABLE
- Query features
  - SELECT-FROM-WHERE statements, set and bag operations, table expressions, aggregation and grouping
  - Subqueries: not much more expressive power added

Next: modifications

---

INSERT

- Insert one row
  - INSERT INTO Enroll VALUES (456, 'CPS216');
    - Student 456 takes CPS216
  - Insert the result of a query
    - INSERT INTO Enroll
      (SELECT SID, 'CPS216' FROM Student
       WHERE SID NOT IN (SELECT SID FROM Enroll
            WHERE CID = 'CPS216'));
      - Force everybody to take CPS216

---

DELETE

- Delete everything
  - DELETE FROM Enroll;
- Delete according to a WHERE condition
  - Example: Student 456 drops CPS216
    - DELETE FROM Enroll
      WHERE SID = 456 AND CID = 'CPS216';
  - Example: Drop students with GPA lower than 1.0 from all CPS classes
    - DELETE FROM Enroll
      WHERE SID IN (SELECT SID FROM Student
            WHERE GPA < 1.0)
      AND CID LIKE 'CPS%';

---

UPDATE

- Example: Student 142 changes name to “Barney” and GPA to 3.0
  - UPDATE Student
    SET name = 'Barney', GPA = 3.0
    WHERE SID = 142;
  - Example: Let’s be “fair”?
    - UPDATE Student
      SET GPA = (SELECT AVG(GPA) FROM Student);
      - But update of every row causes average GPA to change!
      - Average GPA is computed over the old Student table

---

Summary of SQL features covered so far

- Basic modeling features
  - Bags, NULL's
- Schema features
  - CREATE/DROP TABLE
- Query features
  - SELECT-FROM-WHERE statements, set and bag operations, table expressions, aggregation and grouping, subqueries

---

Constraints

- Restrictions on allowable data in a database
  - In addition to the simple structure and type restrictions imposed by the table definitions
  - Declared as part of the schema
  - Enforced automatically by the DBMS
- Why use constraints?
  - Protect data integrity (catch errors)
  - Tell the DBMS about the data (so it can optimize better)
Types of SQL constraints

- **NOT NULL**
- **Key**
- **Referential integrity (foreign key)**
- **General assertion**
- **Tuple- and attribute-based CHECK’s**

NOT NULL constraint examples

- `CREATE TABLE Student
  (SID INTEGER NOT NULL,
   name VARCHAR(30) NOT NULL,
   email VARCHAR(30),
   age INTEGER,
   GPA FLOAT);`
- `CREATE TABLE Course
  (CID CHAR(10) NOT NULL,
   title VARCHAR(100) NOT NULL);`
- `CREATE TABLE Enroll
  (SID INTEGER NOT NULL,
   CID CHAR(10) NOT NULL);`

Key declaration

- **At most one PRIMARY KEY per table**
  - Typically implies a primary index
  - Rows are stored inside the index, typically sorted by the primary key value
- **Any number of UNIQUE keys per table**
  - Typically implies a secondary index
  - Pointers to rows are stored inside the index

Key declaration examples

- `CREATE TABLE Student
  (SID INTEGER NOT NULL PRIMARY KEY,
   name VARCHAR(30) NOT NULL,
   email VARCHAR(30) UNIQUE,
   age INTEGER,
   GPA FLOAT);`
- `CREATE TABLE Course
  (CID CHAR(10) NOT NULL PRIMARY KEY,
   title VARCHAR(100) NOT NULL);`
- `CREATE TABLE Enroll
  (SID INTEGER NOT NULL,
   CID CHAR(10) NOT NULL,
   PRIMARY KEY(SID, CID));`

Referential integrity example

- **Enroll.SID references Student.SID**
  - If an SID appears in Enroll, it must appear in Student
- **Enroll.CID references Course.CID**
  - If a CID appears in Enroll, it must appear in Course
  - That is, no “dangling pointers”

Referential integrity in SQL

- Referenced column(s) must be PRIMARY KEY
- Referencing column(s) form a FOREIGN KEY
- Example
  ```sql
  CREATE TABLE Enroll
  (SID INTEGER NOT NULL
   REFERENCES Student(SID),
   CID CHAR(10) NOT NULL,
   PRIMARY KEY(SID, CID),
   FOREIGN KEY CID REFERENCES Course(CID));
  ```
Enforcing referential integrity

Example: `Enroll.SID` references `Student.SID`
- Insert/update an `Enroll` row so it refers to a non-existent SID  
  ▪ Reject
- Delete/update a `Student` row whose SID is referenced by some `Enroll` row  
  ▪ Reject  
  ▪ Cascade: ripple changes to all referring rows  
  ▪ Set NULL: set all references to NULL
- Deferred constraint checking (e.g., only at the end of a transaction)  
  ▪ Good for performance (e.g., during bulk loading)  
  ▪ Required when creating cycles of references

General assertion

- `CREATE ASSERTION assertion_name CHECK assertion_condition;`
- `assertion_condition` is checked for each modification that could potentially violate it
- Example: `Enroll.SID` references `Student.SID`
  - `CREATE ASSERTION EnrollStudentRefIntegrity CHECK (NOT EXISTS (SELECT * FROM Enroll WHERE SID NOT IN (SELECT SID FROM Student)));`

In SQL3, but not all (perhaps no) DBMS support it

Tuple- and attribute-based CHECK’s

- Associated with a single table
- Only checked when a tuple or an attribute is inserted or updated
- Example:
  - `CREATE TABLE Enroll (SID INTEGER NOT NULL CHECK (SID IN (SELECT SID FROM Student)), CID ...);`
  - Is it a referential integrity constraint?  
  - Not quite; not checked when `Student` is modified

Summary of SQL features covered so far

- Basic modeling features  
  ▪ Bags, NULL’s
- Schema features  
  ▪ `CREATE/DROP TABLE`, constraints  
  ▪ Next: views
- Query features  
  ▪ `SELECT` statements, set and bag operations, table expressions, aggregation and grouping, subqueries
- Modifications

Views

- A view is like a “virtual” table  
  ▪ Defined by a query, which describes how to compute the view contents on the fly  
  ▪ DBMS stores the view definition query instead of view contents  
  ▪ Can be used in queries just like a regular table

Creating and dropping views

- Example: CPS216 roster  
  - `CREATE VIEW CPS216Roster AS SELECT SID, name, age, GPA ...`  
  - Called “base tables”  
  - `FROM Student`  
  - `WHERE SID IN (SELECT SID FROM Enroll WHERE CID = 'CPS216');`

- To drop a view  
  - `DROP VIEW view_name;`
Using views in queries

- Example: find the average GPA of CPS216 students
  - SELECT AVG(GPA) FROM CPS216Roster;
  - To process the query, replace the reference to the view by its definition
  - SELECT AVG(GPA) FROM (SELECT SID, name, age, GPA FROM Student WHERE SID IN (SELECT SID FROM Enroll WHERE CID = 'CPS216'));

Why use views?

- To hide data from users
- To hide complexity from users
- Logical data independence
  - If applications deal with views, we can change the underlying schema without affecting applications
  - Recall physical data independence: change the physical organization of data without affecting applications
- Real database applications use tons of views

Summary of SQL features covered so far

- Basic modeling features
  - Bags, NULL's
- Schema features
  - CREATE/DROP TABLE, constraints, views
- Query features
  - SELECT-FROM-WHERE statements, set and bag operations, table expressions, aggregation and grouping, subqueries
- Modifications
- Next: indexes

Indexes

- An index is an auxiliary persistent data structure
  - Search tree (e.g., B+-tree), lookup table (e.g., hash table), etc.
- More on indexes in following weeks!
- An index on \( R.A \) can speed up accesses of the form
  - \( R.A = value \)
  - \( R.A > value \) (depending on the index type)
- An index on \( (R.A_1, \ldots, R.A_n) \) can speed up
  - \( R.A_1 = value_1 \land \ldots \land R.A_n = value_n \)
  - Multidimensional range searches (depending on the index type)
- Is an index on \( (R.A, R.B) \) equivalent to one on \( (R.B, R.A) \)?
- Or an index on \( R.A \) plus another index on \( R.B \)?

Examples of using indexes

- SELECT * FROM Student WHERE name = 'Bart'
  - Without an index on Student.name: must scan the entire table if we store Student as a flat file of unordered rows
  - With index: go "directly" to rows with name = 'Bart'
- SELECT * FROM Student, Enroll WHERE Student.SID = Enroll.SID;
  - Without any index: for each Student row, scan the entire Enroll table for matching SID
  - With index: go "directly" to Enroll rows with matching SID

Creating and dropping indexes in SQL

- CREATE INDEX index_name ON table_name(column_name, ..., column_name);
- DROP INDEX index_name;

- Typically, the DBMS will automatically create indexes for PRIMARY KEY and UNIQUE constraint declarations
Choosing indexes to create

More indexes = better performance?
- Indexes take space
- Indexes need to be maintained when data is updated
- Indexes have one more level of indirection
  - Perhaps not a problem for main memory, but can be really bad on disk
- Optimal index selection depends on both query and update workload and the size of tables
  - Automatic index selection is still an area of active research

Summary of SQL features covered so far

- Basic modeling features
  - Bags, NULL’s
- Schema features
  - CREATE/DROP TABLE, constraints, views
- Query features
  - SELECT-FROM-WHERE statements, set and bag operations, table expressions, aggregation and grouping, subqueries
- Modifications
- Performance tuning features
  - Indexes

What else?

- Output ordering (ORDER BY)
- Triggers
- SQL transactions and isolation levels
- Application programming interface
- Recursion