Announcements

- Homework #1 due tonight (Sept. 9)
  - Sample solution will be available next Tuesday
- Sign up for student presentations
  - Optional; "replaces" your lowest homework grade
  - Two slots: last lecture before midterm and last lecture before final
  - Three students per slot

SQL

- SQL: Structured Query Language
  - Pronounced “S-Q-L” or “sequel”
  - The standard query language support by most commercial DBMS
- A brief history
  - IBM System R
  - ANSI SQL89
  - ANSI SQL92 (SQL2)
  - SQL3 (still under construction after years!)
Creating and dropping tables

- **CREATE TABLE** `table_name` (...,
  `column_name` `column_type`, ...);
- **DROP TABLE** `table_name`;

**Examples**

```sql
create table Student (SID integer,
  name varchar(30), email varchar(30),
  age integer, GPA float);
create table Course (CID char(10), title varchar(100));
create table Enroll (SID integer, CID char(10));
drop table Student;
drop table Course;
drop table Enroll;
```

-- lines starting with -- are ignored.
-- SQL is insensitive to white space.
-- SQL is case insensitive; writing ...Course... is equivalent to
-- writing ...COURSE...

---

Basic queries: SFW statement

- **SELECT** `A_1`, `A_2`, ..., `A_n`
  FROM `R_1`, `R_2`, ..., `R_m`
  WHERE `condition`;
- Also called an SPJ (select-project-join) query
- Equivalent (not really!) to relational algebra query
  \[
  \pi_{A_1, A_2, ..., A_n} (\sigma_{condition} (R_1 \times R_2 \times ... \times R_m))
  \]

---

Example: reading a table

- **SELECT** `*` FROM `Student`;
  - Single-table query, so no cross product here
  - WHERE clause is optional
  - `*` is a short hand for "all columns"
Example: selection and projection

- Name of students under 18
  - SELECT name FROM Student WHERE age < 18;

- When was Lisa born?
  - SELECT 2004 - age
    FROM Student
    WHERE name = 'Lisa';

- SELECT list can contain expressions
  - Can also use built-in functions such as SUBSTR, ABS, etc.

- String literals (case sensitive) are enclosed in single quotes

Example: join

- SID's and names of students taking courses with the word 'Database' in their titles
  - SELECT Student.SID, Student.name
    FROM Student, Enroll, Course
    WHERE Student.SID = Enroll.SID
    AND Enroll.CID = Course.CID
    AND title LIKE '%Database%';

- LIKE matches a string against a pattern
  - % matches any sequence of 0 or more characters

- Okay to omit table_name in table_name.column_name if column_name is unique

Example: rename

- SID's of all pairs of classmates
  - Relational algebra query:
    \[ \pi_{e1.SID, e2.SID} (\rho_{e1.SID < e2.SID} (\pi_{e1.CID = e2.CID, e1.SID > e2.SID} (\rho_{e1.SID > e2.SID} (\rho_{e1.SID < e2.SID} (e1, e2)))))) \]

- SQL:
  - SELECT e1.SID AS SID1, e2.SID AS SID2
    FROM Enroll AS e1, Enroll AS e2
    WHERE e1.CID = e2.CID
    AND e1.SID > e2.SID;

- AS keyword is completely optional
A more complicated example

- Titles of all courses that Bart and Lisa are taking together

```sql
SELECT c.title
FROM Student sb, Student sl, Enroll eb, Enroll el, Course c
WHERE sb.name = 'Bart' AND sl.name = 'Lisa'
AND eb.SID = sb.SID AND el.SID = sl.SID AND eb.CID = el.CID
AND el.CID = c.CID;
```

Tip: Write the FROM clause first, then WHERE, and then SELECT

Why SFW statements?

- Out of many possible ways of structuring SQL statements, why did the designers choose SELECT-FROM-WHERE?
  - A large number of queries can be written using only selection, projection, and cross product (or join)
  - Any query that uses only these operators can be written in a canonical form: $\pi_L (\sigma_p (R_1 \times \ldots \times R_n))$
    - Example: $\pi_{A,B} (\sigma_A (R) \bowtie_{A,B} \sigma_B (T)) = \pi_{A,B} (R \bowtie_{A,B} T)$
    - $\pi_{A,B} (R \bowtie_{A,B} T) = \pi_{A,B} (R \times S \times T)$
  - SELECT-FROM-WHERE captures this canonical form

Set versus bag semantics

- Set
  - No duplicates
  - Relational model and algebra use set semantics
- Bag
  - Duplicates allowed
  - Number of duplicates is significant
  - SQL uses bag semantics by default
Set versus bag example

\[ \pi_{\text{sid}} \text{Enroll} \]

\[
\begin{array}{c}
\text{SID} \\
142 \\
142 \\
123 \\
857 \\
857 \\
456 \\
\ldots
\end{array}
\]

\[
\begin{array}{c}
\text{CID} \\
\text{CPS196} \\
\text{CPS196} \\
\text{CPS114} \\
\text{CPS196} \\
\text{CPS130} \\
\text{CPS114} \\
\ldots
\end{array}
\]

A case for bag semantics

- Efficiency
  - Saves time of eliminating duplicates
- Which one is more useful?
  - \( \pi_{\text{gpa}} \text{Student} \)
  - SELECT GPA FROM Student;

- Besides, SQL provides the option of set semantics with \textit{DISTINCT} keyword

Operational semantics of SFW

- SELECT (DISTINCT) \( E_1, E_2, \ldots, E_s \)
  FROM \( R_1, R_2, \ldots, R_m \)
  WHERE \textit{condition};
- For each \( t_1 \) in \( R_1 \):
  - For each \( t_2 \) in \( R_2 \): \ldots
  - For each \( t_m \) in \( R_m \):
    - If \textit{condition} is true over \( t_1, t_2, \ldots, t_m \):
      - Compute and output \( E_1, E_2, \ldots, E_s \) as a row
- If \textit{DISTINCT} is present
  - Eliminate duplicate rows in output
- \( t_1, t_2, \ldots, t_m \) are often called tuple variables
Example: forcing set semantics

- SID's of all pairs of classmates
  - SELECT e1.SID AS SID1, e2.SID AS SID2
    FROM Enroll AS e1, Enroll AS e2
    WHERE e1.CID = e2.CID
    AND e1.SID > e2.SID;
  - SELECT DISTINCT e1.SID AS SID1, e2.SID AS SID2
    ...  
    - With DISTINCT, all duplicate (SID1, SID2) pairs are removed
      from the output

SQL set and bag operations

- UNION, EXCEPT, INTERSECT
  - Set semantics
    - Duplicates in input tables, if any, are first eliminated
    - Exactly like set \( \cup \), \( - \), and \( \cap \) in relational algebra
  - UNION ALL, EXCEPT ALL, INTERSECT ALL
    - Bag semantics
    - Think of each row as having an implicit count (the number of times it appears in the table)
    - Bag union: sum up the counts from two tables
    - Bag difference: proper-subtract the two counts
    - Bag intersection: take the minimum of the two counts

Examples of bag operations
Examples of set versus bag operations

- `Enroll(SID, CID), ClubMember(club, SID)`
  - `(SELECT SID FROM ClubMember) EXCEPT (SELECT SID FROM Enroll);`
  - `(SELECT SID FROM ClubMember) EXCEPT ALL (SELECT SID FROM Enroll);`

Summary of SQL features covered so far

- `SELECT-FROM-WHERE` statements (select-project-join queries)
- Set and bag operations
- Next: how to nest SQL queries

Table expression

- Use query result as a table
  - In set and bag operations, `FROM` clauses, etc.
  - A way to "nest" queries
- Example: names of students who are in more clubs than classes
  ```sql
  SELECT DISTINCT name
  FROM Student, ((SELECT SID FROM ClubMember) EXCEPT ALL (SELECT SID FROM Enroll)) AS S
  WHERE Student.SID = S.SID;
  ```
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');
  ```
- Runtime error if the subquery returns more than one row

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 name = 'Bart'
  AND age = s.age);
  ```
  - It is a correlated subquery—a subquery that references tuple variables in surrounding queries
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 s
WHERE EXISTS
  (SELECT * FROM Enroll e
   WHERE [SID] = s.SID
   AND EXISTS
     (SELECT * FROM Enroll
      WHERE [SID] = [s.SID]
      AND CID <> e.CID));
Quantified subqueries

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

*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
    (GPA < ANY (SELECT GPA FROM Student));
  - Use NOT to negate a condition

More ways of getting the highest GPA

- Which students have the highest GPA?
Summary of SQL features covered so far

- SELECT-FROM-WHERE statements
- Set and bag operations
- Table expressions, subqueries
  - Subqueries allow queries to be written in more declarative ways (recall the highest GPA query)
  - But they do not add much expressive power
    - Try translating other forms of subqueries into [NOT] EXISTS, which in turn can be translated into join (and difference)

Next: aggregation and grouping

Aggregates

- Standard SQL aggregate functions: COUNT, SUM, AVG, MIN, MAX
- Example: number of students under 18, and their average GPA
  - SELECT COUNT(*), AVG(GPA)
    FROM Student
    WHERE age < 18;
  - COUNT(*) counts the number of rows

Aggregates with DISTINCT

- Example: How many students are taking classes?
  - SELECT COUNT(DISTINCT SID)
    FROM Enroll;
  - is equivalent to:
**GROUP BY**

- SELECT ... FROM ... WHERE ... 
  GROUP BY list_of_columns;

- Example: find the average GPA for each age group
  - SELECT age, AVG(GPA) FROM Student GROUP BY age;

---

**Operational semantics of GROUP BY**

SELECT ... FROM ... WHERE ... GROUP BY ...;

- Compute FROM (×)
- Compute WHERE (σ)
- Compute GROUP BY: group rows according to the values of GROUP BY columns
- Compute SELECT for each group (π)
  - Number of groups = number of rows in the final output

---

**Example of computing GROUP BY**

SELECT age, AVG(GPA) FROM Student GROUP BY age;

- Compute GROUP BY: group rows according to the values of GROUP BY columns

<table>
<thead>
<tr>
<th>SID</th>
<th>Name</th>
<th>Age</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart</td>
<td>10</td>
<td>2.3</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td>8</td>
<td>4.3</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>10</td>
<td>3.1</td>
</tr>
<tr>
<td>456</td>
<td>Ralph</td>
<td>8</td>
<td>2.3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Compute SELECT for each group

<table>
<thead>
<tr>
<th>Age</th>
<th>AVG_GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2.7</td>
</tr>
<tr>
<td>8</td>
<td>3.0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Aggregates with no GROUP BY

- An aggregate query with no GROUP BY clause represent a special case where all rows go into one group

SELECT AVG(GPA) FROM Student;

Restriction on SELECT

- If a query uses aggregation/group by, then every column referenced in SELECT must be either
  - Aggregated, or
  - A GROUP BY column
- This restriction ensures that any SELECT expression produces only one value for each group

Examples of invalid queries

- SELECT MAX(age) FROM Student GROUP BY age;
  - Recall there is one output row per group
  - There can be multiple SID values per group
- SELECT MAX(GPA) FROM Student;
  - Recall there is only one group for an aggregate query with no GROUP BY clause
  - There can be multiple SID values
  - Wishful thinking (that the output SID value is the one associated with the highest GPA) does NOT work
HAVING

- Used to filter groups based on the group properties (e.g., aggregate values, \textit{GROUP BY} column values)
- \textbf{SELECT ... FROM ... WHERE ... GROUP BY ...}
  \textit{HAVING} \textit{condition};
  - Compute \textit{FROM} (\times)
  - Compute \textit{WHERE} (\sigma)
  - Compute \textit{GROUP BY}: group rows according to the values of \textit{GROUP BY} columns
  - Compute \textit{HAVING} (another \sigma over the groups)
  - Compute \textit{SELECT} for each group that passes the \textit{HAVING} condition (\Pi)

HAVING examples

- Find the average GPA for each age group over 10
  - \textbf{SELECT age, AVG(GPA)}
    \textit{FROM} Student
    \textit{GROUP BY age}
    \textit{HAVING} age > 10;
  - Can be written using \textit{WHERE} without table expressions
- List the average GPA for each age group with more than a hundred students
  - \textbf{SELECT age, AVG(GPA)}
    \textit{FROM} Student
    \textit{GROUP BY age}
    \textit{HAVING} COUNT(*) > 100;
  - Can be written using \textit{WHERE} and table expressions

Summary of SQL features covered so far

- \textbf{SELECT-FROM-WHERE statements}
- Set and bag operations
- Table expressions, subqueries
- Aggregation and grouping
  - More expressive power than relational algebra

- Next: ordering output rows
ORDER BY

▷ SELECT [DISTINCT] ...
FROM ... WHERE ... GROUP BY ... HAVING ...
ORDER BY output_column (ASC | DESC), ...;
▷ ASC = ascending, DESC = descending
▷ Operational semantics
  ▷ After SELECT list has been computed and optional
duplicate elimination has been carried out,
sort the output according to ORDER BY specification

ORDER BY example

▷ List all students, sort them by GPA (descending)
and name (ascending)
  ▷ SELECT SID, name, age, GPA
     FROM Student
     ORDER BY GPA DESC, name;
  ▷ ASC is the default option
  ▷ Strictly speaking, only output columns can appear in
   ORDER BY clause (although some DBMS support more)
  ▷ Can use sequence numbers of output columns instead
   ORDER BY 4 DESC, 2;

Summary of SQL features covered so far

▷ SELECT-FROM-WHERE statements
▷ Set and bag operations
▷ Table expressions, subqueries
▷ Aggregation and grouping
▷ Ordering

▷ Next: NULL’s, outerjoins, data modification, constraints, …