
Lecture 4: SQL

Dragan Mirkovic
Department of Computer Science
University of Houston

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Announcements

- Quiz #2 Today
 - Ch. 4 (Relational algebra and Calculus)
- Questions?
- Quiz #3
 - Ch. 5, SQL (5.1 – 5.5)

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Review

Exercise 4.3 Consider the following schema:

Suppliers(sid: integer, sname: string, address: string)
Parts(pid: integer, pname: string, color: string)
Catalog(sid: integer, pid: integer, cost: real)

1. Find the *names* of suppliers who supply some red part.
2. Find the *sids* of suppliers who supply some red or green part.
3. Find the *sids* of suppliers who supply some red part or are at 221 Packer Street.
4. Find the *sids* of suppliers who supply some red part and some green part.

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Answers

1. $\pi_{sname}(\pi_{sid}((\pi_{pid}\sigma_{color='red'} Parts) \bowtie Catalog) \bowtie Suppliers)$
2. $\pi_{sid}(\pi_{pid}(\sigma_{color='red'} \vee \sigma_{color='green'} Parts) \bowtie catalog)$
3. $\rho(R1, \pi_{sid}((\pi_{pid}\sigma_{color='red'} Parts) \bowtie Catalog))$
 $\rho(R2, \pi_{sid}\sigma_{address='221PackerStreet'} Suppliers)$
 $R1 \cup R2$
4. $\rho(R1, \pi_{sid}((\pi_{pid}\sigma_{color='red'} Parts) \bowtie Catalog))$
 $\rho(R2, \pi_{sid}((\pi_{pid}\sigma_{color='green'} Parts) \bowtie Catalog))$
 $R1 \cap R2$

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Review ...

Exercise 4.4 Consider the Supplier-Parts-Catalog schema from the previous question. State what the following queries compute:

1. $\pi_{sname}(\pi_{sid}((\sigma_{color='red'} Parts) \bowtie (\sigma_{cost < 100} Catalog)) \bowtie Suppliers)$

2. $\pi_{sname}(\pi_{sid}((\sigma_{color='red'} Parts) \bowtie (\sigma_{cost < 100} Catalog) \bowtie Suppliers))$

1. Find the Supplier names of the suppliers who supply a red part that costs less than 100 dollars.
2. This Relational Algebra statement does not return anything because of the sequence of projection operators. Once the sid is projected, it is the only field in the set. Therefore, projecting on sname will not return anything.

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Introduction

- **Structured Query Language (SQL)** is the most widely used commercial relational database language
- It includes:
 - DML (Data Manipulation Language)
 - DDL (Data Definition Language)
 - Triggers and Advanced Integrity Constraints
 - Embedded and Dynamic SQL (calls from C and other lang.)
 - Client-Server Execution and Remote access
 - Transaction management
 - Security (Views etc.)
 - Advanced features

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Example Instances

- We will use these instances of the Sailors and Reserves relations in our examples.
- If the key for the Reserves relation contained only the attributes *sid* and *bid*, how would the semantics differ?

R1

<u>sid</u>	<u>bid</u>	<u>day</u>
22	101	10/10/96
58	103	11/12/96

S1

<u>sid</u>	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Basic SQL Query

```
SELECT [DISTINCT] target-list
FROM relation-list
WHERE qualification
```

- relation-list** A list of relation names (possibly with a range-variable after each name).
- target-list** A list of attributes of relations in relation-list
- qualification** Comparisons of the form
 $Attr\ op\ const\ OR\ Attr1\ op\ Attr2,$
 – where *op* is one of (<, >, =, ≤, ≥, ≠) combined using AND, OR and NOT.
- DISTINCT** is an optional keyword indicating that the answer should not contain duplicates.
- Default is that duplicates are not eliminated!

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Conceptual Evaluation Strategy

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
 - Compute the cross-product of relation-list.
 - Discard resulting tuples if they fail qualifications.
 - Delete attributes that are not in target-list.
 - If **DISTINCT** is specified, eliminate duplicate rows.
- This strategy is probably the least efficient way to compute a query! An optimizer will find more efficient strategies to compute the same answers.
- Without **DISTINCT** the result is a **multiset** of rows
 - Unordered collection of elements with possible multiple copies of some elements

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Example of Conceptual Evaluation

- Find the names of sailors who have reserved boat number 103.

```
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid=R.sid AND R.bid=103
```

- For simplicity use shorter instances of S and R :

<u>sid</u>	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

<u>sid</u>	<u>bid</u>	<u>day</u>
22	101	10/10/96
58	103	11/12/96

Instances S4 of Sailors and R3 of reserves

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Solution

- 1st step: construct the cross-product of S4 x R3

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

- 2nd step: Apply qualification `s.sid=R.sid AND R.bid=103`
- 3rd step: Eliminate unwanted columns

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

A Note on Range Variables

- The range variables are optional
- Really needed only if the same relation appears twice in the FROM clause (to resolve the ambiguity)
- The previous query can also be written as:

```
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid=R.sid AND bid=103
```

OR

```
SELECT sname
FROM Sailors, Reserves
WHERE Sailors.sid=Reserves.sid AND bid=103
```

*It is good style,
however, to use
range variables
always!*

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Example

- Find `sid` of sailors who've reserved at least one boat

```
SELECT S.sid
FROM   Sailors S, Reserves R
WHERE  S.sid=R.sid
```

- Would adding `DISTINCT` to this query make a difference?
- What is the effect of replacing `S.sid` by `S.sname` in the `SELECT` clause?
- Would adding `DISTINCT` to this variant of the query make a difference?

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Expressions and Strings

```
SELECT S.age, age1=S.age-5, 2*S.age AS age2
FROM   Sailors S
WHERE  S.sname LIKE 'B_%B'
```

- Illustrates use of arithmetic expressions and string pattern matching: Find triples (of ages of sailors and two fields defined by expressions) for sailors whose names begin and end with B and contain at least three characters.
- **AS** and **=** are two ways to name fields in result.
- **LIKE** is used for string matching.
 - `'_'` stands for any one character and
 - `'%'` stands for 0 or more arbitrary characters.

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Set Operations in SQL

- The result of a query is a multiset of rows
 - One can introduce set operations like
 - Union, intersection and difference
 - Implemented as **UNION**, **INTERSECT** and **EXCEPT**
 - Other set operations
 - IN** – check if an element is in a given set
 - EXISTS** – check if set is empty
 - NOT** – prefix for **IN** and **EXISTS**
 - Set comparison operators
 - op **ANY** and op **ALL** – compare the values with elements in the set

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Example

- Find sid's of sailors who've reserved a red or a green boat
- **UNION**: Can be used to compute the union of any two union-compatible sets of tuples (which are themselves the result of SQL queries).

```
SELECT S.sid
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.bid=B.bid
      AND (B.color='red' OR B.color='green')
```

```
SELECT S.sid
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.bid=B.bid
      AND B.color='red'
```
- If we replace **OR** by **AND** in the first version, what do we get?

```
UNION
SELECT S.sid
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.bid=B.bid
      AND B.color='green'
```
- Also available: **EXCEPT** (What do we get if we replace

UNION by EXCEPT?
D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Example

- Find sid's of sailors who've reserved a red and a green boat

- INTERSECT**: Can be used to compute the intersection of any two union-compatible sets of tuples.

```
SELECT S.sid
FROM   Sailors S, Boats B1,
       Reserves R1, Boats B2, Reserves R2
WHERE  S.sid=R1.sid AND R1.bid=B1.bid
       AND S.sid=R2.sid AND R2.bid=B2.bid
       AND B1.color='red'
       AND B2.color='green'
```

- Included in the SQL/92 standard, but some systems don't support it.
- Contrast symmetry of the **UNION** and **INTERSECT** queries with how much the other versions differ.

```
SELECT S.sid ← Key field!
FROM   Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid
       AND B.color='red'
```

```
INTERSECT
SELECT S.sid
FROM   Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid AND
       B.color='green'
```

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

First Summary SQL

- User expresses “what he wants” without worrying “how it is implemented” (this is the job of the query optimizer).
- SQL is set oriented; operations can be defined on sets (or multisets) rather than having to specify loops (as it was the case with the predecessors of SQL).
- More features to express integrity constraints and triggers to support data driven programming have been added more recently.
- Efforts are under way to make SQL to become a “full programming language” → PL/SQL, SQL3.

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Nested Queries

- A very powerful feature of SQL:
 - **WHERE** clause can itself contain an SQL query! (called subquery)
 - The same is true for **FROM** and **HAVING** clauses.
- **Example:** *Find names of sailors who've reserved boat #103:*

```
SELECT S.sname
FROM Sailors S
WHERE S.sid IN (SELECT R.sid
                FROM Reserves R
                WHERE R.bid=103)
```
- To find sailors who've not reserved #103, use NOT IN.
- To understand semantics of nested queries, think of a nested loops evaluation: For each Sailors tuple, check the qualification by computing the subquery.

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Examples

- We can also write multiply nested queries
 - *Find the names of sailors who have reserved a red boat*

```
SELECT S.sname
FROM Sailors S
WHERE S.sid IN (SELECT R.sid
                FROM Reserves R
                WHERE R.bid IN (SELECT B.bid
                                FROM Boats B
                                WHERE B.color='red'))
```
 - How to find the names of sailors who have not reserved a red boat?

```
WHERE S.sid NOT IN (...)
```
 - How to find the names of sailors who have reserved a boat that is not red?

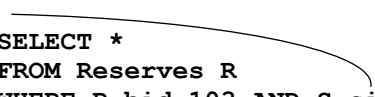
```
WHERE R.bid NOT IN (...)
```
 - How to find the names of sailors who have not reserved a boat that is not red?

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Nested Queries with Correlation

- The inner queries so far have been completely independent of the outer query
- *Find names of sailors who've reserved boat #103:*

```
SELECT S.sname
FROM Sailors S
WHERE EXISTS (SELECT *
              FROM Reserves R
              WHERE R.bid=103 AND S.sid=R.sid)
```


- **EXISTS** is another set comparison operator
 - Test whether the set is nonempty
- The subquery depends on S and needs to be re-evaluated for each row of S
 - *Correlated queries*

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Correlated Nested Queries...

- If **EXISTS** is replaced by **NOT EXISTS**
 - Names of sailors who have not reserved a red boat
- **UNIQUE** predicate can be used with **EXISTS**
 - returns true if no row appears twice in the answer to the subquery
- Example:
 - find sailors with at most one reservation for boat #103.
- If **UNIQUE** is used, and * is replaced by **R.bid**, finds sailors with at most one reservation for boat #103. (**UNIQUE** checks for duplicate tuples; * denotes all attributes. Why do we have to replace * by R.bid?)
- Illustrates why, in general, subquery must be re-computed for each Sailors tuple.

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Set-Comparison Operators

- We've already seen **IN**, **EXISTS** and **UNIQUE**.
- We can also use **NOT IN**, **NOT EXISTS** and **NOT UNIQUE**.
- Also available: **op ANY**, **op ALL**, where **op** is one of
 $>, <, =, \geq, \leq, \neq$

- Example:

- Find sailors whose rating is greater than that of some sailor called Horatio:

```
SELECT *
FROM Sailors S
WHERE S.rating > ANY (SELECT S2.rating
                     FROM Sailors S2
                     WHERE S2.sname='Horatio')
```

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Rewriting INTERSECT Queries Using IN

Find sid's of sailors who've reserved both a red and a green boat:

```
SELECT S.sid
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.bid=B.bid AND B.color='red'
      AND S.sid IN (SELECT S2.sid
                   FROM Sailors S2, Boats B2, Reserves R2
                   WHERE S2.sid=R2.sid AND R2.bid=B2.bid
                   AND B2.color='green')
```

- Similarly, **EXCEPT** queries re-written using **NOT IN**.
- To find names (not sid's) of Sailors who've reserved both red and green boats, just replace **S.sid** by **S.sname** in **SELECT** clause. (What about **INTERSECT** query?)

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Division in SQL

- Find sailors who've reserved all boats.
- Let's do it the hard way, without **EXCEPT**:
 - For each sailor we check if there is no boat that has not been reserved by this sailor

```
SELECT S.sname
FROM Sailors S
WHERE NOT EXISTS (SELECT B.bid
                  FROM Boats B
                  WHERE NOT EXISTS (SELECT R.bid
                                    FROM Reserves R
                                    WHERE R.bid=B.bid
                                    AND R.sid=S.sid))
```

- There are two correlations in the innermost query

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Solution with EXCEPT

- For each sailor S check if there is a boat that S has not reserved

```
SELECT S.sname
FROM Sailors S
WHERE NOT EXISTS ((SELECT B.bid
                   FROM Boats B)
                  EXCEPT
                  (SELECT R.bid
                   FROM Reserves R
                   WHERE R.sid=S.sid))
```

- Steps:
 1. Select all `bid` in `B`
 2. Eliminate those that `sid` has reserved
 3. The empty set in a result means that `sid` has reserved all boats

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Aggregate Operators

- Significant extension of relational algebra
 - Perform operations on the resulting multisets
- Operators:
 - COUNT ([DISTINCT] A)
 - the number of (unique) values in the A column
 - SUM ([DISTINCT] A)
 - the sum of all (unique) values in the A column
 - AVG ([DISTINCT] A)
 - the average of all (unique) values in the A column
 - MAX (A)
 - MIN (A)
- Examples:

```
SELECT COUNT (*)
FROM Sailors S

SELECT AVG (S.age)
FROM Sailors S
WHERE S.rating=10

SELECT COUNT (DISTINCT S.rating)
FROM Sailors S
WHERE S.sname='Bob'

SELECT AVG (DISTINCT S.age)
FROM Sailors S
WHERE S.rating=10

SELECT S.name, S.age
FROM Sailors S
WHERE S.rating=(SELECT MAX(S2.rating)
                FROM Sailors S2)
```

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Examples

- *Find name and age of the oldest sailor(s)*
- We could try the following query

```
SELECT S.sname, MAX (S.age)
FROM Sailors S
```

 - This query is illegal! Aggregates in **SELECT** cannot be mixed with non-aggregates unless **GROUP BY** is used
- We have to use a nested query

```
SELECT S.sname, S.age
FROM Sailors S
WHERE S.age = (SELECT MAX(S2.age)
              FROM Sailors S2)

SELECT S.sname, S.age
FROM Sailors S
WHERE (SELECT MAX(S2.age)
      FROM Sailors S2) = S.age
```
- The third query is equivalent to the second query, and is allowed in the SQL/92 standard, but is not supported in some systems.

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Remark

- The effect of aggregate operators is reduction of a relation column to a single number
 - The output is not a relation!
- Aggregation is an operation that cannot be expressed in relational algebra!

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

GROUP BY and HAVING

- So far, we've applied aggregate operators to all (qualifying) tuples. Sometimes, we want to apply them to each of several groups of tuples.
- Consider: Find the age of the youngest sailor for each rating level.
 - In general, we don't know how many rating levels exist, and what the rating values for these levels are!
 - Suppose we know that rating values go from 1 to 10; we can write 10 queries that look like this (!):

```
For  $i = 1, 2, \dots, 10$ :  
    SELECT MIN (S.age)  
    FROM   Sailors S  
    WHERE  S.rating =  $i$ 
```

- It requires a major extension to the basic SQL query form
 - **GROUP BY** clause with optional **HAVING** clause

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Queries With GROUP BY and HAVING

```
SELECT [DISTINCT] select-list
FROM      relation-list
WHERE     qualification
GROUP BY grouping-list
HAVING   group-qualification
```

- The *select-list* contains
 - attribute list and terms with aggregate operations (e.g., `MIN (S.age)`).
- The *attribute list* must be a subset of *grouping-list!*.
- Intuitively, each answer tuple corresponds to a *group*, and these attributes must have a single value per group.
- A *group* is a set of tuples that have the same value for all attributes in *grouping-list*.)

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Conceptual Evaluation

- The cross-product of relation-list is computed, tuples that fail qualification are discarded, 'unnecessary' fields are deleted, and the remaining tuples are partitioned into groups by the value of attributes in grouping-list.
- The group-qualification is then applied to eliminate some groups. Expressions in group-qualification **must have a single value per group!**
 - In effect, an attribute in group-qualification that is not an argument of an aggregate op also appears in grouping-list. (SQL does not exploit primary key semantics here!)
- **One answer tuple is generated per qualifying group.**

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Solution

- Find the age of the youngest sailor for each rating level.

- Using **GROUP BY** clause we can write

```
SELECT S.rating, MIN(S.age)
FROM Sailors S
GROUP BY S.rating
```

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Example

- Find the age of the youngest sailor with age ≥ 18 , for each rating with at least 2 such sailors

```
SELECT S.rating, MIN (S.age) AS minage
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating
HAVING COUNT (*) > 1
```

- Only S.rating and S.age are mentioned in the **SELECT**, **GROUP BY** or **HAVING** clauses
 - other attributes 'unnecessary'.
- 2nd column of result is unnamed.
 - Use **AS** to name it.

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Solution

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
71	zorba	10	16.0
64	horatio	7	35.0
29	brutus	1	33.0
58	rusty	10	35.0

Instance S3 of Sailors

rating	age
7	35.0

Answer relation

Steps:

1. Compute cross product
2. Apply **WHERE** clause
3. Eliminate unwanted columns
4. Sort columns according to the **GROUP BY** clause

rating	age
1	33.0
7	45.0
7	35.0
8	55.5
10	35.0

5. Apply **GROUP BY** qualification

Example

- For each red boat, find the number of reservations for this boat:

```
SELECT B.bid, COUNT (*) AS scout
FROM Boats B, Reserves R
WHERE R.bid=B.bid AND B.color='red'
GROUP BY B.bid
```

- Grouping over a join of two relations.
- What do we get if we remove B.color='red' from the **WHERE** clause and add a **HAVING** clause with this condition?

Answer

- This query is illegal:

```
SELECT B.bid, COUNT (*) AS scount
FROM Boats B, Reserves R
WHERE R.bid=B.bid
GROUP BY B.bid
HAVING B.color = 'red'
```

- Only columns that appear in the **GROUP BY** clause can appear in the **HAVING** clause unless they are arguments of aggregate operators.

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Example

- Find the age of the youngest sailor with age > 18, for each rating with at least 2 sailors (of any age)

```
SELECT S.rating, MIN (S.age)
FROM Sailors S
WHERE S.age > 18
GROUP BY S.rating
HAVING 1 < (SELECT COUNT (*)
            FROM Sailors S2
            WHERE S.rating=S2.rating)
```

- Shows HAVING clause can also contain a subquery.
- Compare this with the query where we considered only ratings with 2 sailors over 18!
- What if HAVING clause is replaced by:
 - HAVING COUNT(*) >1

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Example

- Find those ratings for which the average age is the minimum over all ratings
- Aggregate operations cannot be nested! WRONG:

```
SELECT S.rating
FROM Sailors S
WHERE S.age = (SELECT MIN(AVG(S2.age))
              FROM Sailors S2)
```
- Correct solution (SQL/92)

```
SELECT Temp.rating, Temp.avgage
FROM (SELECT S.rating, AVG(S.age) AS avgage
      FROM Sailors S
      GROUP BY S.rating) AS Temp
WHERE Temp.avgage = (SELECT MIN(Temp.avgage)
                    FROM Temp)
```
- IN SQL:1999 two new set functions: **EVERY** and **ANY**
 - Similar to **WHERE** clause

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Null Values

- Field values in a tuple are sometimes *unknown* (e.g., a rating has not been assigned) or *inapplicable* (e.g., no spouse's name).
 - SQL provides a special value null for such situations.
- The presence of null complicates many issues. E.g.:
 - Special operators needed to check if value is/is not null.
 - Is rating>8 true or false when rating is equal to null? What about AND, OR and NOT connectives?
 - We need a 3-valued logic (true, false and unknown).
 - Meaning of constructs must be defined carefully. (e.g., WHERE clause eliminates rows that don't evaluate to true.)
 - New operators (in particular, outer joins) possible/needed.
 - SQL provides a special comparison operator **IS NULL**

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Nulls and Logical Expressions

- Short circuit evaluation of logical expressions
 - (true) OR (unknown) = (true)
 - (false) AND (unknown) = (false)
- Otherwise the result of expressions involving **unknown** **is unknown**
 - NOT (unknown) = (unknown)

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Integrity Constraints (Review)

- An IC describes conditions that every legal instance of a relation must satisfy.
 - Inserts/deletes/updates that violate IC's are disallowed.
 - Can be used to ensure application semantics (e.g., sid is a key), or prevent inconsistencies (e.g., sname has to be a string, age must be < 200)
- Types of IC's: Domain constraints, primary key constraints, foreign key constraints, general constraints.
 - Domain constraints: Field values must be of right type. Always enforced.

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

General Constraints

- Useful when more general ICs than keys are involved.
- Can use queries to express constraint.
- Constraints can be named.

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Examples

```
CREATE TABLE Sailors
  (sid INTEGER,
   sname CHAR(10),
   rating INTEGER,
   age REAL,
   PRIMARY KEY (sid),
   CHECK (rating >= 1 AND rating <= 10 ))
```

Table constraint form:
CHECK conditional-expression

Table constraint:

- Rating must be an integer between 1 and 10.

```
CREATE TABLE Reserves
  (sname CHAR(10),
   bid INTEGER,
   day DATE,
   PRIMARY KEY (bid, day),
```

Table constraint:

- Interlake boats cannot be reserved

```
CONSTRAINT noInterlakeRes
CHECK ('Interlake' <> (SELECT B.bname
                       FROM Boats B
                       WHERE B.bid=bid)))
```

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

Constraints Over Multiple Relations

- Table constraints are associated with a single table
 - CHECK clause can refer to other tables
 - Required to hold *only* if the associated table is not empty!
 - Example:
 - *Number of boats plus number of sailors is < 100*
- ```
CREATE TABLE Sailors
(sid INTEGER, sname CHAR(10),
 rating INTEGER, age REAL,
 PRIMARY KEY (sid),
 CHECK ((SELECT COUNT (S.sid) FROM Sailors S)
 + (SELECT COUNT (B.bid) FROM Boats B) < 100)
```
- Awkward and wrong!
  - If Sailors is empty, the number of Boats tuples can be anything!

---

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

## ASSERTIONS

---

- IC over several tables
- **ASSERTION** is the right solution
  - not associated with either table.

```
CREATE ASSERTION smallClub
CHECK ((SELECT COUNT (S.sid) FROM Sailors S)
 + (SELECT COUNT (B.bid) FROM Boats B) < 100)
```

---

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

# Triggers

---

- Trigger: procedure that starts automatically if specified changes occur to the DBMS
- Typically specified by the DBA
- **Active database** = database + set of triggers
- Description of triggers has three parts:
  - Event (activates the trigger)
  - Condition (tests whether the triggers should run)
  - Action (what happens if the trigger runs)
- A trigger works as a 'daemon' that monitors a database for specified events
- Events: insert, delete, update...
- Condition: true/false statement

---

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

# Trigger syntax

---

- Trigger syntax depends on the implementation
- Oracle server syntax

```
CREATE TRIGGER init_count BEFORE INSERT ON Students
DECLARE
 count INTEGER
BEGIN
 count:=0;
END
CREATE TRIGGER incr_count AFTER INSERT ON Students
WHEN (new.age < 18)
FOR EACH ROW
BEGIN
 count:= count + 1;
END
```

---

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

## Triggers: Example (SQL:1999)

---

- Compare to the SQL: 1999 syntax

```
CREATE TRIGGER youngSailorUpdate AFTER INSERT ON SAILORS
REFERENCING NEW TABLE NewSailors
FOR EACH STATEMENT
 INSERT
 INTO YoungSailors(sid, name, age, rating)
 SELECT sid, name, age, rating
 FROM NewSailors N
 WHERE N.age <= 18
```

- **NEW TABLE** specifies the name for the set of new tuples
- **FOR EACH STATEMENT** specifies a statement-level trigger

---

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

## Designing Active Databases

---

- Triggers add a powerful mechanism for dealing with changes to a database
- The effects can be very complex
  - If there is more than one trigger for the same event DBMS executes them in arbitrary order!
  - Execution of the trigger can activate another trigger
  - Chain activation
  - Recursive triggers
  - Maintenance can become very difficult
  - Sometimes it is safer and simpler to use integrity constraints only
- Applications of triggers:
  - Alert users to unusual events
  - Maintenance of an events log

---

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

## Summary

---

- SQL was an important factor in the early acceptance of the relational model; more natural than earlier, procedural query languages.
- Relationally complete; in fact, significantly more expressive power than relational algebra.
- Even queries that can be expressed in RA can often be expressed more naturally in SQL.
- Many alternative ways to write a query; optimizer should look for most efficient evaluation plan.
  - In practice, users need to be aware of how queries are optimized and evaluated for best results.

---

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

## Summary (Contd.)

---

- NULL for unknown field values brings many complications
- Embedded SQL allows execution within a host language; cursor mechanism allows retrieval of one record at a time
- APIs such as ODBC and ODBC introduce a layer of abstraction between application and DBMS
- SQL allows specification of rich integrity constraints
- Triggers respond to changes in the database

---

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

# Homework

---

- Read Ch 5.
- Page 174, 5.1(1-6, 11), 5.3 (2, 3, 5), 5.4 (3, 6)

---

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

## Exercise 5.1

---

Student(*snum*: integer, *sname*: string, *major*: string, *level*: string, *age*: integer)

Class(*name*: string, *meets\_at*: string, *room*: string, *fid*: integer)

Enrolled(*snum*: integer, *cname*: string)

Faculty(*fid*: integer, *fname*: string, *deptid*: integer)

Write the following queries in SQL. No duplicates should be printed in any of the answers.

1. Find the names of all Juniors (level = JR) who are enrolled in a class taught by I. Teach.
2. Find the age of the oldest student who is either a History major or enrolled in a course taught by I. Teach.
3. Find the names of all classes that either meet in room R128 or have five or more students enrolled.
4. Find the names of all students who are enrolled in two classes that meet at the same time.
5. Find the names of faculty members who teach in every room in which some class is taught.
6. Find the names of faculty members for whom the combined enrollment of the courses that they teach is less than five.
11. Find the names of students not enrolled in any class.

---

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

# Solutions

---

```
SELECT DISTINCT S.Sname
FROM Student S, Class C, Enrolled E, Faculty F
WHERE S.snum = E.snum AND E.cname = C.name AND
 C.fid = F.fid AND F.fname = 'I.Teach' AND S.level = 'JR'

SELECT MAX(S.age)
FROM Student S
WHERE (S.major = 'History')
 OR S.snum IN (SELECT E.snum
 FROM Class C, Enrolled E, Faculty F
 WHERE E.cname = C.name AND C.fid = F.fid
 AND F.fname = 'I.Teach')
```

```
SELECT C.name
FROM Class C
WHERE C.room = 'R128'
 OR C.name IN (SELECT E.cname
 FROM Enrolled E
 GROUP BY E.cname
 HAVING COUNT (*) >= 5)
```

---

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005

# Solutions

---

```
SELECT DISTINCT S.sname
FROM Student S
WHERE S.snum IN
 (SELECT E1.snum
 FROM Enrolled E1, Enrolled E2, Class C1, Class C2
 WHERE E1.snum = E2.snum AND E1.cname <> E2.cname
 AND E1.cname = C1.name
 AND E2.cname = C2.name AND C1.meets at = C2.meets at)
```

```
SELECT DISTINCT F.fname
FROM Faculty F
WHERE NOT EXISTS ((SELECT *
 FROM Class C)
 EXCEPT
 (SELECT C1.room
 FROM Class C1
 WHERE C1.fid = F.fid))
```

---

D. Mirkovic, COSC 3480: Design of File and Database Systems, Spring 2005