

## Chapter 5: Structured Query Language (SQL) part the second

The rest of Chapter 5

## New Students Example

- Student(snum,sname,major,level,age)
- Class(name,meets\_at,room,fid)
- Enrolled(snum,cname)
- Faculty(fid,fname,deptid)

30

## Class

Name	Meets_at	Room	FID
Data Structures	MwF 10	R128	489456522
Database Systems	MwF 12:30-1:45	1320 DCL	142519864
Operating System Design	TuTh 12-1:20	20 AW	489456522
Archaeology of the Incas	MwF 3-4:15	R128	248965255
Aviation Accident Investigation	TuTh 1-2:50	Q3	011564812
Air Quality Engineering	TuTh 10:30-11:45	R15	011564812
Introductory Latin	MwF 3-4:15	R12	248965255
American Political Parties	TuTh 2-3:15	20 AW	619023588
Social Cognition	Tu 6:30-8:40	R15	159542516
Perception	MTuWTh 3	Q3	489221823
Multivariate Analysis	TuTh 2-3:15	R15	090873519
Patent Law	F 1-2:50	R128	090873519
Urban Economics	MwF 11	20 AW	489221823
Organic Chemistry	TuTh 12:30-1:45	R12	489221823
Marketing Research	Mw 10-11:15	1320 DCL	489221823
Seminar in American Art	M 4	R15	489221823
Orbital Mechanics	MwF 8 1320	DCL	011564812
Dairy Herd Management	TuTh 12:30-1:45	R128	356187925
Communication Networks	Mw 9:30-10:45	20 AW	141582651
Optical Electronics	TuTh 12:30-1:45	R15	254099823
Introduction to Math	TuTh 8-9:30	R128	489221823

31

## Student

SNUM	SNAME	MAJOR	ST	AGE
51135593	Maria White	English	SR	21
60839453	Charles Harris	Architecture	SR	22
99354543	Susan Martin	Law	JR	20
112348546	Joseph Thompson	Computer Science	SO	19
115987938	Christopher Garcia	Computer Science	JR	20
132977562	Angela Martinez	History	SR	20
269734834	Thomas Robinson	Psychology	SO	18
280158572	Margaret Clark	Animal Science	FR	18
301221823	Juan Rodriguez	Psychology	JR	20
318548912	Dorothy Lewis	Finance	FR	18
320874981	Daniel Lee	Electrical Engineering	FR	17
322654189	Lisa Walker	Computer Science	SO	17
348121549	Paul Hall	Computer Science	JR	18
351565322	Nancy Allen	Accounting	JR	19
451518064	Mark Young	Finance	FR	18
455798411	Luis Hernandez	Electrical Engineering	FR	17
462156489	Donald King	Mechanical Engineering	SO	19
550156548	George Wright	Education	SR	21
552455318	Ana Lopez	Computer Engineering	SR	19
556784565	Kenneth Hill	Civil Engineering	SR	21
567354612	Karen Scott	Computer Engineering	FR	18
573284895	Steven Green	Kinesiology	SO	19
574489456	Betty Adams	Economics	JR	20
578875478	Edward Baker	Veterinary Medicine	SR	21

32

## Enrolled

SNUM	CNAME
112348546	Database Systems
115987938	Database Systems
348121549	Database Systems
322654189	Database Systems
552455318	Database Systems
455798411	Operating System Design
552455318	Operating System Design
567354612	Operating System Design
112348546	Operating System Design
115987938	Operating System Design
322654189	Operating System Design
567354612	Data Structures
552455318	Communication Networks
455798411	Optical Electronics
455798411	Organic Chemistry
301221823	Perception
301221823	Social Cognition
301221823	American Political Parties
556784565	Air Quality Engineering
99354543	Patent Law
574489456	Urban Economics

33

## Faculty

FID	FNAME	DEPTID
142519864	I. Teach	20
242518965	James Smith	68
141582651	Mary Johnson	20
11564812	John Williams	68
254099823	Patricia Jones	68
356187925	Robert Brown	12
489456522	Linda Davis	20
287221212	Michael Miller	12
248965255	Barbara Wilson	12
159542516	William Moore	33
90873519	Elizabeth Taylor	11
486512566	David Anderson	20
619023588	Jennifer Thomas	11
489221823	Richard Jackson	33
548977562	Ulysses Teach	20

34

## What kinds of queries can you answer so far?

- Find the names of all classes taught by Elizabeth Taylor
- Find the departments that have more than one faculty member
- Find the student ids of those who have taken a course named “Database Systems”

35

## Trickier question

- What are the student ids of those who have taken a course with “Database” in the name?

36

## A string walks into a bar...

```
SELECT DISTINCT sid
FROM   enrolled e, class c
WHERE  e.cname = c.name
       AND c.name LIKE '%Database%'
```

- LIKE is used for string matching:
  - ‘\_’ stands for any one character and
  - ‘%’ stands for 0 or more arbitrary characters.
- To match “Strange%”, need an escape character:  
**like ‘Strange%’ escape ‘\’**
- SQL supports string operations such as
  - concatenation (using “||”)
  - converting from upper to lower case (and vice versa)
  - finding string length, extracting substrings, etc.

37

## An expression’s worth 1000 words

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

- Illustrates arithmetic expressions and string pattern matching
  - Looks 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.

38

## Ordering of Tuples

- List in alphabetic order the names of sailors who reserved a red boat

```
select distinct sname
from   Sailors, Boats, Reserves
where  Sailors.sid = Reserves.sid and
       Boats.bid = Reserves.bid and
       colour='red'
order by sname
```

- Order is specified by:
  - desc for descending order
  - asc for ascending order (default)
  - E.g. **order by sname desc**

39

## Set Operations

- **union**, **intersect**, and **except** correspond to the relational algebra operations  $\cup$ ,  $\cap$ ,  $-$ .
- Each automatically eliminates duplicates; To retain all duplicates use the corresponding multiset versions:
  - union all**, **intersect all** and **except all**.
- Suppose a tuple occurs  $m$  times in  $r$  and  $n$  times in  $s$ , then, it occurs:
  - $m + n$  times in  $r$  **union all**  $s$
  - $\min(m, n)$  times in  $r$  **intersect all**  $s$
  - $\max(0, m - n)$  times in  $r$  **except all**  $s$

40

## Find sid's of sailors who've reserved a red or a green boat

- **UNION:** Can union any two *union-compatible* sets of tuples (i.e., 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.colour='red' OR
      B.colour='green')
```
- Also available: **EXCEPT** (What if we replace UNION by EXCEPT?)  

```
SELECT S.sid
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.bid=B.bid
AND B.colour='red'
```
- Some systems use **MINUS** for EXCEPT.
- Replace OR by AND in the first version,  

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

41

## Set Operations: Intersect

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

```
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.colour='red' AND
       B2.colour='green')
```
- **INTERSECT:** Can be used to compute the intersection of any two *union-compatible* sets of tuples.  

```
SELECT S.sid
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.bid=B.bid
AND B.colour='red'
```
- In SQL/92, but some systems don't support it.  

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

42

## What can you do now?

- Find the sids of all students who took Operating System Design but did not take Databases

43

## But what about...

- Select the IDs of all students who have not taken "Operating System Design"

44

## Nested Queries

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 bid=103)
```

- A very powerful feature of SQL: a **WHERE** clause can itself contain an SQL query! (Actually, so can FROM clauses.)
- To find sailors who've *not* reserved #103, use **NOT IN**.
- To understand nested query semantics, think of a *nested loops* evaluation:
  - For each Sailors tuple, check the qualification by computing the subquery.

45

## Okay, so when would you *really* want to use nesting

- One good hint: not in
- E.g., "Find me names of sailors who have *not* reserved boat #103"

46

## Nested Queries with Correlation

Find names of sailors who've reserved boat #103:

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

- **EXISTS**: returns true if the set is not empty.
- **UNIQUE**: returns true if there are no duplicates.
- If **UNIQUE** is used above, and \* is replaced by *bid*, finds sailors with at most one reservation for boat #103.
  - (\* denotes all attributes. Why do we have to replace \* by *bid*?)
- Illustrates why, in general, subquery must be re-computed for each Sailors tuple.

47

## More on Set-Comparison Operators

- We've already seen **IN**, **EXISTS** and **UNIQUE**. Can also use **NOT IN**, **NOT EXISTS** and **NOT UNIQUE**.
- Also available: **op ANY**, **op ALL**, where op is one of: **>**, **<**, **=**, **<=**, **>=**, **<>**
- Find sailors whose rating is greater than that of some sailor called "rusty":

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

48

## 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.colour='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.colour='green')
```

- Similarly, **EXCEPT** queries can be 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?)

49

## Division in SQL

Find sailors who've reserved all boats.

```
(2) SELECT 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))
```

The hard way (without **EXCEPT**):

```
(1) SELECT 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))
```

select sailor S such that ...

there is no boat B...

which is not reserved by S

50

## Example

- Find the name and age of the oldest student(s)

51

## You're Now Leaving the World of Relational Algebra

- You now have many ways of asking relational algebra queries
  - For this class, you should be able write queries using all of the different concepts that we've discussed & know the terms used
  - In general, use whatever seems easiest
  - Sometimes the query optimizer may do poorly, and you'll need to try a different version, but we'll ignore that for this class.

52

## Mind the gap

- But there's more you might want to know!
- E.g., "find the average age of sailors"
- There are extensions of Relational Algebra that cover these topics
  - We won't cover them

53

## Aggregate Operators

- These functions operate on the multiset of values of a column of a relation, and return a value
  - AVG**: average value
  - MIN**: minimum value
  - MAX**: maximum value
  - SUM**: sum of values
  - COUNT**: number of values
- The following versions eliminate duplicates before apply the operation to attribute A:
  - COUNT ( DISTINCT A)**
  - SUM ( DISTINCT A)**
  - AVG ( DISTINCT A)**

54

## Aggregate Operators: Examples

```
SELECT COUNT (*)
FROM Sailors

SELECT sname
FROM Sailors S
WHERE S.rating= (SELECT MAX(S2.rating)
FROM Sailors S2)

SELECT AVG (age)
FROM Sailors
WHERE rating=10

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

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

55

## Aggregate Operators: Examples(cont)

### Find name and age of the oldest sailor(s)

- The first query is illegal! (We'll see the reason later, when we discuss **GROUP BY**.)

```
SELECT sname, MAX (age)
FROM Sailors
```
- Second query is OK: can use value = subquery only if subquery returns single value.

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

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

56

## Aggregation examples

- Find the average student age
- How many students have taken a class with "Database" in the title

57

## GROUP BY and HAVING

- Divide tuples into groups and apply aggregate operations to each group.
- Example: *Find the age of the youngest sailor for each rating level.*
  - Suppose rating values go from 1 to 10; we can write 10 queries that look like this (!):

```
For i = 1, 2, ..., 10: SELECT MIN (age)
FROM Sailors
WHERE rating = i
```
- Problem:  
We don't know how many rating levels exist, and what the rating values for these levels are!

58

## GROUP BY and HAVING (cont)

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

- The *target-list* contains
  - attribute names
  - terms with aggregate operations (e.g., MIN (S.age)).
- Attributes in (i) must also be in *grouping-list*.
  - each answer tuple corresponds to a *group*.
  - *group* = a set of tuples with same value for all attributes in *grouping-list*
  - selected attributes must have a single value per group.
- Attributes in *group-qualification* are either in *grouping-list* or are arguments to an aggregate operator. 59

## GROUP BY and HAVING (cont)

- **Example1:** For each boat, find the age of the youngest sailor who has reserved this boat:

```
SELECT bid, MIN (age)
FROM Sailors S, Reserves R
WHERE S.sid= R.sid
GROUP BY bid
```

- **Example2:** For each boat with more than 10 reservations, find the age of the youngest sailor who has reserved this boat:

```
SELECT bid, MIN (age)
FROM Sailors S, Reserves R
WHERE S.sid= R.sid
GROUP BY bid
HAVING COUNT (*) > 10 ← per group qualification! 60
```

## Grouping Examples

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

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

- Only rating and age are mentioned *alone* in the SELECT, GROUP BY or HAVING clauses.
- 2nd column of result is unnamed. (Use AS to name it.)

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

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

*Answer relation* 61

## Conceptual Evaluation of a Query

- compute the cross-product of *relation-list*
- keep only tuples that satisfy *qualification*
- partition the remaining tuples into groups by the value of attributes in *grouping-list*
- keep only the groups that satisfy *group-qualification* (expressions in *group-qualification* must have a single value per group!)
- delete fields that are not in *target-list*
- generate one answer tuple per qualifying group. 62

## Groupies of your very own

- Find the average age for each class standing (e.g., Freshman)
- Find the deptID and # of faculty members for each department having an id > 20
- Find the deptID and # of faculty members for each department with > 2 faculty

63

## Grouping Examples (cont')

For each red boat, find the number of sailors who reserved it

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

- Grouping over a join of three relations.
- What if we:
  - remove *B.colour='red'* from the WHERE clause, and then
  - add a HAVING clause with the dropped condition?
- What if we replace COUNT (DISTINCT R.sid) with COUNT (\*) ? 64

## Grouping Examples (cont')

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.
- What if HAVING clause is replaced by:
  - HAVING COUNT(\*) > 1

65

## Grouping Examples (cont')

Find those ratings for which their average age is the minimum over all ratings

```
SELECT S.rating
FROM   Sailors S
WHERE  S.age = (SELECT MIN (AVG (S2.age)) FROM Sailors S2)
```

- **WRONG!** Aggregate operations cannot be nested!

- **Correct solution (in SQL/92 and SQL/99)**

```
SELECT Temp.rating, Temp.avgage
FROM   (SELECT S.rating, AVG (S.age) avgage
        FROM   Sailors S
        GROUP BY S.rating) AS Temp
WHERE  Temp.avgage = (SELECT MIN (Temp.avgage)
                    FROM   Temp)
```

66

## Views

- Hide data from certain users. To create a view use:  
**CREATE VIEW** *vname* **AS** <query expression>

where:

- <query expression> is any legal expression
- *vname* is the view name

- Example: Suppose tables  
Course(Course#, title, dept)  
Enrolled(Course#, sid, mark)

```
CREATE VIEW CourseWithFails(dept, course#, mark) AS
SELECT  C.dept, C.course#, mark
FROM    Course C, Enrolled E
WHERE   C.dept = E.dept AND
        C.course# = E.course# AND mark < 50
```

This view gives the dept, course#, and marks for those courses where someone failed

67

## Views and Security

- Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s).
  - Given CourseWithFails, but not Course or Enrolled, we can find the course in which some students failed, but we can't find the students who failed.

68

## View Updates

- View updates must occur at the base tables.
  - Ambiguous
  - Difficult
- DBMS's restrict view updates only to some simple views on single tables (called updatable views)
- How to handle DROP TABLE if there's a view on the table?
- DROP TABLE command has options to prevent a table from being dropped if views are defined on it:
  - DROP TABLE Student RESTRICT
    - drops the table, unless there is a view on it
  - DROP TABLE Student CASCADE
    - drops the table, and recursively drops any view referencing it

69

## With Clause

- Allows views to be defined locally to a query, rather than globally.

- Example:

```
WITH BRReserves(sid, bid, colour, date) AS
SELECT sid, bid, colour, date
FROM   Boat B, Reserves R
WHERE  B.bid = R.bid AND (colour = 'red'
                        OR colour = 'blue')
```

```
SELECT sid, name
FROM   Sailors S, BRReserves R
WHERE  S.sid = R.sid AND date > '1/1/03'
```

70

## The Beauty of Views

Find the ratings for whose average age is the minimum over all ratings

- Without views:

```
SELECT Temp.rating, Temp.avgage
FROM (SELECT S.rating, AVG (S.age) avgage
      FROM Sailors S
      GROUP BY S.rating) AS Temp
WHERE Temp.avgage = (SELECT MIN (Temp.avgage)
                    FROM Temp)
```

- With views:

```
With Temp(rating,avgage) as
SELECT S.rating, AVG (S.age) AS avgage
FROM Sailors S
GROUP BY S.rating
Select Temp.rating, Temp.avgage
From Temp
Temp.avgage = (SELECT MIN(Temp.avgage) from Temp)
```

71

## Null Values

- Tuples may have a null value, denoted by *null*, for some of their attributes
- Value *null* signifies an unknown value or that a value does not exist.
- The predicate **IS NULL** ( **IS NOT NULL** ) can be used to check for null values.
  - E.g. Find all sailor names whose age is not known.

```
SELECT name
FROM Sailors
WHERE age IS NULL
```
- The result of any arithmetic expression involving *null* is *null*
  - E.g.  $5 + null$  returns *null*.

72

## Null Values and Three Valued Logic

- null requires a 3-valued logic using the truth value *unknown*:
  - OR: (*unknown* or *true*) = *true*, (*unknown* or *false*) = *unknown*, (*unknown* or *unknown*) = *unknown*
  - AND: (*true* and *unknown*) = *unknown*, (*false* and *unknown*) = *false*, (*unknown* and *unknown*) = *unknown*
  - NOT: (*not unknown*) = *unknown*
  - "*P* is *unknown*" evaluates to true if predicate *P* evaluates to *unknown*
- Any comparison with *null* returns *unknown*
  - E.g.  $5 < null$  or  $null <= null$  or  $null = null$
- Result of **where** clause predicate is treated as *false* if it evaluates to *unknown*
- All aggregate operations except **count(\*)** ignore tuples with null values on the aggregated attributes.

73

## More fun with joins

- What happens if I execute query:

```
Select *
From student s, enrolled e
Where s.snum = e.snum
```
- To get *all* students, you need an *outer join*
- There are several special joins declared in the *from* clause:
  - Inner join – default: only include matches
  - Left outer join – include all tuples from left hand relation
  - Right outer join – include all tuples from right hand relation
  - Full outer join – include all tuples from both relations
- Orthogonal: can have natural join (as in relational algebra)
- Oracle uses (+) after side you want to pad out

Example: 

```
SELECT *
FROM Student S NATURAL LEFT OUTER JOIN Enrolled E
```

74

## Database Manipulation Insertion redux

- Can insert a single tuple using:

```
INSERT INTO Student
VALUES (53688, 'Smith', '222 W.15th ave', 333-4444, MATH)
```
- or

```
INSERT INTO Student (sid, name, address, phone, major)
VALUES (53688, 'Smith', '222 W.15th ave', 333-4444, MATH)
```
- Add a tuple to student with null address and phone:

```
INSERT INTO Student (sid, name, address, phone, major)
VALUES (33388, 'Chan', null, null, CPSC)
```

75

## Database Manipulation Insertion redux (cont)

- Can add values selected from another table
- Make a reservation for sailor 22 with every red boat for 1/1/03

```
INSERT INTO Reserves
SELECT 22, bid, '1/1/03'
FROM Boats
WHERE colour = 'red'
```

- The select-from-where statement is fully evaluated before any of its results are inserted  
So, queries like

```
INSERT INTO table1 SELECT * FROM table1
```

are ok.

76

## Database Manipulation Deletion

- Note that only whole tuples are deleted.
- Can delete all tuples satisfying some condition (e.g., name = Smith):
 

```
DELETE FROM Student
WHERE name = 'Smith'
```
- Delete all sailors whose age is above the average sailor age:
 

```
DELETE FROM Sailors
WHERE age > (SELECT avg(age)
FROM Sailors)
```

77

## Database Manipulation Updates

- Increase the rating of all sailors by 2 (should not be more than 10)
 

```
UPDATE Sailors
SET rating = 10
WHERE rating >= 8
```
- Need to write two updates:
 

```
UPDATE Sailors
SET rating = rating + 2
WHERE rating < 8
```
- Is the order important?

78

## 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 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

79

## General Constraints: Check

- Created with a CHECK clause.
  - Constraints can be named
  - Can use subqueries to express constraint
- ```
CREATE TABLE Sailors
(sid INTEGER,
sname CHAR(10),
rating INTEGER,
age REAL,
PRIMARY KEY (sid),
CHECK (rating >= 1
AND rating <= 10));

CREATE TABLE Reserves
(sname CHAR(10),
bid INTEGER,
day DATE,
PRIMARY KEY (bid, day),
CONSTRAINT noInterlakeRes
CHECK ('Interlake' <>
(SELECT B.bname
FROM Boats B
WHERE B.bid=bid)));
```

Check constraints are checked when tuples are inserted or modified

80

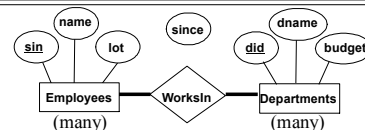
## Domain Constraints

- User can create a new domain and set constraints for it.
- Example:
 

```
CREATE DOMAIN agedomain INTEGER
DEFAULT 21
CHECK (VALUE >= 1 AND VALUE <= 110)
```
- In current systems distinct domains are not distinct types. – i.e., they can be compared
- New systems support distinct types

81

## Constraints over Multiple Relations: Remember this one?



- We couldn't express "every employee works in a department and every department has some employee in it"?
- Neither foreign-key nor not-null constraints in Works\_In can do that.
- Assertions to the rescue!

82

## Constraints Over Multiple Relations

- Cannot be defined in one table.
- Are defined as ASSERTIONS which are not associated with any table
- Example: *Every employee works in one department*

```
CREATE ASSERTION totalEmployment
CHECK
(NOT EXISTS ((SELECT SIN FROM Employee)
EXCEPT
(SELECT SIN FROM WorksIn)));
```

83

## Triggers

- Trigger : a procedure that starts automatically if specified changes occur to the DBMS
- Active Database: a database with triggers
- A trigger has three parts:
  1. Event (activates the trigger)
  2. Condition (tests whether the trigger should run)
  3. Action (procedure executed when trigger runs)
- Database vendors did not wait for trigger standards! So trigger format depends on the DBMS
- NOTE: triggers may cause cascading effects.

84

## Triggers: Example (SQL:1999)

```
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;
```

event

newly inserted tuples

apply once per statement

action

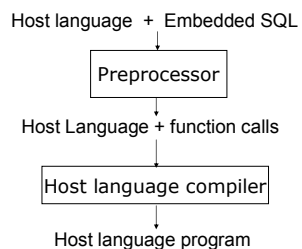
85

## Embedded SQL

- Direct SQL is rarely used: usually, SQL is embedded in some application code.
- We need some method to reference SQL statements.
- But: there is an *impedance mismatch* problem.
- So: we use cursors.
- Many things can be explained with the impedance mismatch.

86

## Programs with SQL



87

## The Impedance Mismatch Problem

The host language manipulates variables, values, pointers

SQL manipulates relations.

There is no construct in the host language for manipulating relations.

Why not use only one language?

- Forgetting SQL: "we can quickly dispense with this idea" [Ullman & Widom, pg. 363].
- SQL cannot do everything that the host language can do.

88

## Embedded SQL: Interface SQL/Host Language

- Values get passed through shared variables
- SQL commands can be called from within a host language (e.g., C++, C or Java) program.
  - SQL statements can refer to host variables (including special variables used to return status).
  - Must include a statement to *connect* to the right database.
- In C/C++, EXEC SQL statement is used to identify embedded SQL request to the preprocessor  
EXEC SQL <embedded SQL statement > ;
- In Java embedding uses  
# SQL { .... } ;

89

## Cursors

- Structures in the host language used to store the results of an SQL query (i.e., solve impedance mismatch)
- In C/C++ can declare a cursor on a relation or query statement (which generates a relation).
- Example:

```
EXEC SQL DECLARE sinfo CURSOR FOR
SELECT S.sname
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.bid=B.bid AND
B.colour='red'
```
- In Java, you'd use an iterator.
- Kinda gross. Instead, how about...

90

## Database APIs: Alternative to embedding

- Rather than modify compiler, add library with database calls (API)
- Special standardized interface: procedures/objects
  - Passes SQL strings from language, presents result sets in a language-friendly way – solves that impedance mismatch
  - Microsoft's *ODBC* is a C/C++ standard on Windows
  - Sun's *JDBC* a Java equivalent
  - API's are DBMS-neutral
    - a "driver" traps the calls and translates them into DBMS-specific code

91

## A glimpse into your possible future: JDBC

- JDBC supports a variety of features for querying and updating data, and for retrieving query results
- JDBC also supports metadata retrieval, such as querying about relations present in the database and the names and types of relation attributes
- Model for communicating with the database:
  - Open a connection
  - Create a "statement" object
  - Execute queries using the Statement object to send queries and fetch results
  - Exception mechanism to handle errors

92

## SQL API in Java (JDBC)

```
Connection con = // connect
DriverManager.getConnection(url, "login", "pass");
Statement stmt = con.createStatement(); // set up stmt
String query = "SELECT name, rating FROM Sailors";
ResultSet rs = stmt.executeQuery(query);
try { // handle exceptions
    // loop through result tuples
    while (rs.next()) {
        String s = rs.getString("name");
        int n = rs.getFloat("rating");
        System.out.println(s + " " + n);
    }
} catch (SQLException ex) {
    System.out.println(ex.getMessage ()
        + ex.getSQLState () + ex.getErrorCode ());
}
```

93

## 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.
- Consists of a data definition, data manipulation and query language.
- 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.

94

## Summary (Cont')

---

- NULL for unknown field values brings many complications
- SQL allows specification of rich integrity constraints (and triggers)
- Embedded SQL allows execution within a host language; cursor mechanism allows retrieval of one record at a time
- APIs such as ODBC and JDBC introduce a layer of abstraction between application and DBMS