SE240: Introduction to Database Systems

Lecture 05:

SQL

Outline

- **Introduction**
- Data Manipulation Language (DML)
  - Simple SQL Queries
  - Nested SQL Queries
  - Aggregation and Grouping
  - Others, More Example: Relational Algebra and SQL
- Advanced Concepts (DDL)
  - Introduction to View
  - Introduction to Assertion/Trigger
  - Introduction to Database Programming

Introduction

- Structured Query Language (SQL) is the most widely used commercial relational database language.
- It was originally developed at IBM in the SEQUEL-XRM and System-R projects (1974-1977).
  - SEQUEL (Structured English Query Language)
- Almost immediately, other vendors introduced DBMS products based on SQL, and it is now a de facto standard.
- SQL continues to evolve in response to the changing need.
  - The current ANSI/ISO standard for SQL is called SQL:1999. The previous standard is SQL-92.
- Not all DBMS products support the full SQL standards.

Introduction

- The SQL language has several aspects to it
  - The Data Manipulation Language (DML)
    - The subset of SQL that allows users to pose queries and to insert, delete, and modify rows.
  - The Data Definition Language (DDL)
    - The subset of SQL that supports the creation, deletion, and modification to definitions for tables and views.
  - Triggers and Advanced Integrity Constraints
    - The new SQL:1999 standard includes support for triggers, which are actions executed by the DBMS whenever changes to the database meet conditions specified in the trigger

Outline

- Introduction
- **Data Manipulation Language (DML)**
  - **Simple SQL Queries**
  - Nested SQL Queries
  - Aggregation and Grouping
  - Others, More Example: Relational Algebra and SQL
- Advanced Concepts (DDL)
  - Introduction to View
  - Introduction to Assertion/Trigger

Retrieval Queries in SQL

- SQL has one basic statement for retrieving information from a database; the SELECT statement
  - This is **NOT** the same as the SELECT (σ) operation of the relational algebra
- Important distinction between SQL and the formal relational model:
  - SQL allows a table (relation) to have two or more tuples that are identical in all their attribute values
  - Hence, an SQL relation (table) is a *multi-set* (sometimes called a bag) of tuples; it is not a set of tuples

Basic SQL Query

- The basic form of SQL

```
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.

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

- qualification
  - Comparisons (*attr op const* or *attr1 op attr2*, where *op* is one of $\geq, \leq, <, =, >, \neq$) combined using AND, OR and NOT.
- DISTINCT
  - an optional keyword indicating that the answer should not contain duplicates.
- Default is that duplicates are not eliminated.

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

- SELECT clause
  - specifies columns to be retained in the result.
- FROM clause
  - specifies a cross-product of tables.
- WHERE clause (optional)
  - specifies selection conditions on the tables mentioned in the FROM clause.
- An SQL query intuitively corresponds to a relational algebra expression involving selections, projections, and cross-products.

SELECT DISTINCT $a_1, a_2, \ldots, a_n$

FROM $R_1, R_2, \ldots, R_m$

WHERE P

$\Pi a_1, a_2, \cdots, a_n (\sigma_P (R_1 \times R_2 \times \cdots \times R_m))$

||

$\Pi_{a1, a2, \ldots, an} (\sigma_P (R_1 \times R_2 \times \ldots \times R_m))$

Example: SELECT-PROJECT

- Find the names of all branches in the loan relation.

```
SELECT branch-name
FROM Loan
```

Loan

branch_name	loan_number	amount
MUST	222	2
CTM	333	1
MUST	777	2

Result

branch_name
MUST
CTM
MUST

Example: DISTINCT Keyword

- Find the names of all branches in the loan relation and remove duplicate.

```
SELECT DISTINCT branch-name
FROM Loan
```

Loan

branch_name	loan_number	amount
MUST	222	2
CTM	333	1
MUST	777	2

Result

branch_name
MUST
CTM

$$\Pi_{branch\_name}(Loan)$$

Basic SQL Query: Processing Steps

only one target list allowed

③	SELECT	[DISTINCT] *target-list*
①	FROM	*relation-list*
②	WHERE	*qualification*

- Conceptual Evaluation Strategy
  - Compute the cross-product of *relation-list*. (Not 'join')
  - 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.)

The Rename Operation in SQL

- Renaming relations and attributes using the ***AS*** clause:

  old-name ***AS*** new-name

- For convenience, usually, “as” can be omitted in FROM clause.

```
SELECT L.branch-name AS name
FROM   Loan AS L
```

- ***AS*** in SQL is used to define a logical variable, called range variable, in the context of either FROM or explicit JOIN.

$\Pi_{sname}(\sigma_{bid=103,\ sailors.sid=reserves.sid}(Sailors \times Reserves)) = \Pi_{sname}(\sigma_{bid=103}(Sailors \bowtie Reserves))$

Find the names of sailors who have reserved boat number 103

Sailors(sid, sname, rating, age)

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

Reserves(sid, bid, day)

sid	bid	day
22	101	10/10/05
58	103	11/12/05

Rename & Cross Product

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

JOIN Condition

Sailors: sid, sname, rating, age

Boats: bid, bname, color

Reserves: sid, bid, day

S.sid	sname	rating	age	R.sid	bid	day
22	dustin	7	45.0	22	101	10/10/05
22	dustin	7	45.0	58	103	11/12/05
31	lubber	8	55.5	22	101	10/10/05
31	lubber	8	55.5	58	103	11/12/05
58	rusty	10	35.0	22	101	10/10/05
58	rusty	10	35.0	58	103	11/12/05

Row remains after selection.

Result

because there will be duplicate sid in the.

$\Pi_{sname}\ \sigma_{bid=103,\ S.sid=R.sid}\ (Sailors \times Reserves)$

$\Pi_{sname}\ \sigma_{bid=103}\ (Sailors \bowtie Reserves)$

The Rename Operation in SQL

- A Note on Range Variables
  - Really needed only if the same relation appears twice in the FROM clause.
- The previous query can also be written as:

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

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

or

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

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
  - `**%**' stands for 0 or more arbitrary characters.

Examples: Simple SQL

- Given the following schema:

Sailors (sid: integer, sname: string, rating:integer, age: real)

Boats (bid: integer, bname: string, color: string)

Reserves (sid: integer, bid: integer, day: date)

Sailors

sid	sname	rating	age

Boats

bid	bname	color

Reserves

sid	bid	day

Examples

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

- Q1: Find the sids of sailors who have reserved a red boat.

$\Pi_{sid}(\sigma_{Boats.bid = Reserves.bid,\ Boats.color = 'red'}(Boats \times Reserves))$

```
SELECT  R.sid
FROM  Boat B, Reserves R
WHERE  B.bid = R.bid AND B.color = 'red'
```

$= \Pi_{sid}(\sigma_{Boats.color = 'red'}(Boats \bowtie Reserves))$

- Q2: Find the names of sailors who have reserved a red boat.

$\Pi_{sname}(\sigma_{Boats.bid = Reserves.bid,\ Sailors.sid = Reserves.sid,\ Boats.color = 'red'}(Sailors \times Boats \times Reserves))$

```
SELECT  S.sname
FROM  Sailors S, Reserves R, Boat B
WHERE  S.sid = R.sid AND R.bid = B.bid AND
          B.color = 'red'
```

$= \Pi_{sname}(\sigma_{Boats.color = 'red'}(Sailors \bowtie Boats \bowtie Reserves))$

Examples

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

- Q3: Find the colors of boats reserved by Lubber.

$\Pi_{color}(\sigma_{Sailors.sid=Reserves.sid, Boats.bid=Reserves.bid, Sailors.name="Lubber"}(Sailors \times Boats \times Reserves))$

```
SELECT  B.color
FROM  Sailors S, Reserves R, Boat B
WHERE  S.sid = R.sid AND R.bid = B.bid AND
          S.name = 'Lubber'.
```

$= \Pi_{color}(\sigma_{Sailors.name="Lubber"}(Sailors \bowtie Boats \bowtie Reserves))$

(In general, there may be more than one sailor called Lubber. In this case, it will return the colors of boats reserved by all such Lubber's).

Examples

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

- Q4: Find the names of sailors who have reserved at least one boat. $\Pi_{sname}(\sigma_{Sailors.sid = Reserves.sid}(Sailors \times Reserves)) = \Pi_{sname}(Sailors \bowtie Reserves)$

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

(If a sailor has not made a reservation, the second step in the conceptual evaluation strategy would eliminate all rows in the cross-product that involve this sailor).

Ordering the Display of Tuples

- The ORDER BY clause is used to sort the tuples in a query result based on the values of some attribute(s)
- For example: List in alphabetic order the names of all sailors

```
SELECT    S.name
FROM      Sailors S
ORDER BY  S.name
```

- ***ASC*** for ascending order (default)
- ***DESC*** for descending order
- SQL must perform a sort to fulfill an ORDER BY request. Since sorting a large number of tuples may be costly, it is desirable to sort only when necessary.

Set Operations

- The set operation UNION, INTERSECT, and EXCEPT operate on relations and correspond to the relational algebra operations $\cup$, $\cap$ and $-$.
- The set operations apply only to union compatible relations.
- Each of the above operations *automatically eliminates duplicates*; to retain all duplicates, we can use 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

Examples

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

- Q5: Find sid's of sailors who've reserved a red *or* a green boat
- **UNION**: compute the union of any two union-compatible sets of tuples (which are themselves the result of SQL queries).
- If we replace **OR** by **AND** in the first version, what do we get?

error!

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

```
SELECT R.sid
FROM Boats B, Reserves R
WHERE R.bid=B.bid
      AND B.color='red'
UNION
SELECT R.sid
FROM Boats B, Reserves R
WHERE R.bid=B.bid
      AND B.color='green'
```

Examples

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

- Q6: Find sid's of sailors who've reserved a red *and* a green boat
- **INTERSECT:** compute the intersection of any two union-compatible sets of tuples.
- Included in the SQL/92 standard, but some systems don't support it.

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

```
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')
```

Examples

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

- Q7: Find sid's of all sailors who've reserved red boat but not green boat.

it indicates that there exists red boats for these sailors.

```
SELECT  R.sid
FROM  Boats B, Reserves R
WHERE  R.bid=B.bid AND B.color='red'

EXCEPT

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

This operation indicates that sailors reserving green boats are also included.

can we create a new table to complete the query ?

Examples

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

- Q8: Find sid's of all sailors who have a rating of 10 *or* reserved boat 104

```
SELECT  S.sid
FROM  Sailor S
WHERE  S.rating = 10

UNION

SELECT  R.sid
FROM  Reserves R
WHERE  R.bid=104
```

Outline

- Introduction
- **Data Manipulation Language (DML)**
  - Simple SQL Queries
  - **Nested SQL Queries**
  - Aggregation and Grouping
  - Others, More Example: Relational Algebra and SQL
- Advanced Concepts (DDL)
  - Introduction to View
  - Introduction to Assertion/Trigger

Nested Queries

- A nested query is a query that has another query embedded within it.
- The embedded query is called a subquery.
- The embedded query can be a nested query itself.
- Every SQL statement returns: a relation/set in the result OR Null OR A single atomic value
- We can replace a value or set of values with a SQL statement (ie., a subquery)
  - Illegal if the subquery returns the wrong type for the comparison
- Queries may have very deeply nested structures.

Nested Queries

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

- Q9: 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)
```

S.sid determines S.sname

⇔

```
FROM Sailor S, Reserves R
WHERE S.sid=R.sid AND
      R.bid=103
```

- A very powerful feature of SQL: a **WHERE** clause can itself contain an SQL query! (Actually, so can FORM clauses.)
- 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.*

Correlated Nested Queries

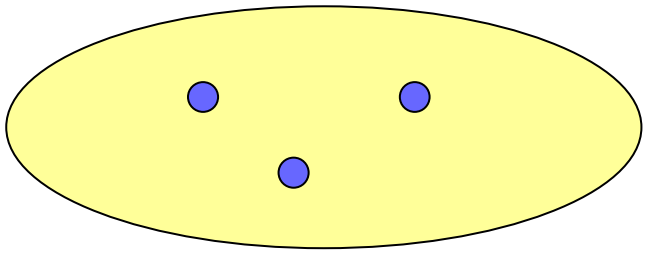
```
SELECT  S.sname
FROM  Sailors S
WHERE  S.sid IN  (SELECT  R.sid
                  FROM  Reserves R
                  WHERE  R.bid=103)
```

- In the previous example, the inner subquery is independent of the outer query.
- In general, the inner subquery could depend on the row currently being examined in the outer query.

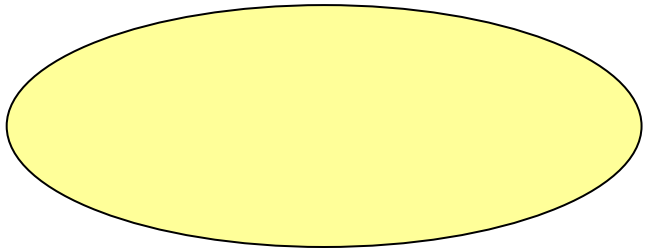
```
SELECT  S.sname
FROM  Sailors S
WHERE  1 <= (SELECT  COUNT(R.sid)
             FROM  Reserves R
             WHERE  R.bid=103 AND R.sid = S.sid)
```

Test for Empty Relations

- Word EXISTS returns TURE if the argument subquery is nonempty.

EXISTS()

TRUE

EXISTS()

FALSE

Examples

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

The 'UNIQUE' constraint ensures that all values in a column are different

- Q9': 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, which allows us to test whether a set is nonempty.
- If **UNIQUE** is used instead of EXISTS, and * is replaced by *R.sid*, *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.sid*?)

duplicate

represents all columns

Since UNIQUE can only test a certain column.

Set-Comparison Operators

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

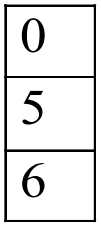
- We've already seen **IN**, **EXISTS** and **UNIQUE**.

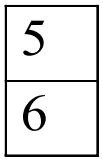
  We can also use **NOT IN**, **NOT EXISTS** and **NOT UNIQUE**.

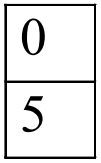
- Also available: ***op* ANY**, ***op* ALL**
  - Where op is one of the arithmetic comparison operator $\geq, \leq, <, =, >, \neq$
  - SOME is also available, but it is just a synonym for ANY.
- Q10: Find sailors whose rating is greater than that of some sailor called Horatio:

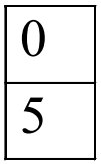
```
SELECT  *
FROM  Sailors S
WHERE  S.rating > ANY (SELECT  S2.rating
                       FROM  Sailors S2
                       WHERE S2.sname='Horatio')
```

SOME/ANY Semantics

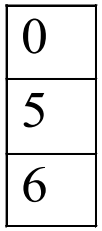
(5 > some ) returns TRUE (5 > 0)

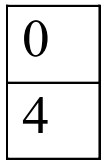
(5 > some ) returns FALSE

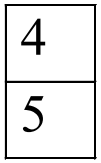
(5 = some ) = TRUE

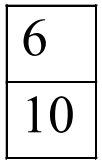
(5 ≠ some ) = TRUE (since 0 ≠ 5)

ALL Semantics

(5 > all ) = FALSE

(5 > all ) = TRUE

(5 = all ) = FALSE

(5 ≠ all ) = TRUE

Examples

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

- Q11: Find sailors whose rating is better than every sailor called Horatio.

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

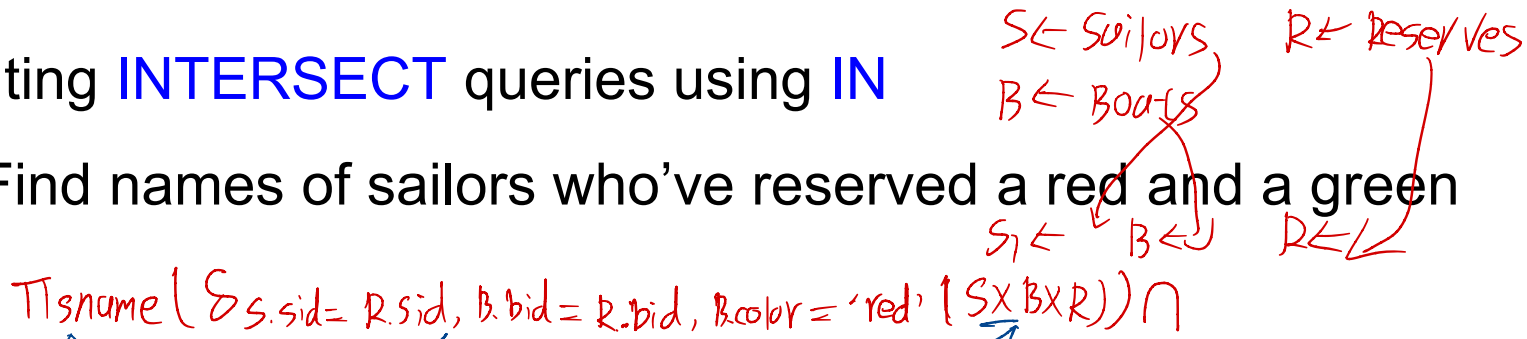
- Q12: Find the sailors with the highest rating.

```
SELECT  *
FROM  Sailors S
WHERE  S.rating >= ALL  (SELECT  S2.rating
                         FROM  Sailors S2)
```

Examples

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

- Rewriting INTERSECT queries using IN
- Q6': Find names of sailors who've reserved a red and a green boat



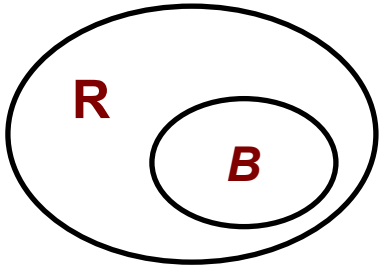

```
SELECT  S.name
FROM  Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid AND B.color='red'
          AND S.sid IN  (SELECT  R2.sid
                         FROM Boats B2, Reserves R2
                         WHERE  R2.bid=B2.bid
                                AND  B2.color='green')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

- Similarly, EXCEPT queries can be re-written using NOT IN.

Division in SQL

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

- Q13: Find sailors who've reserved all boats.



NOT EXIST (B – R)

all boats

$\pi_{B.bid}$ (Boats)

```
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))
```

All boats

All boats reserved by S

- Note that this query is correlated – for each sailor S, we check to see that the set of boats reserved by S includes every boat.
- Logic: there does not exist any boat that is not reserved by S

A Working Example

Sailors

sid	sname	rating	age
s01	Tom	10.2	26
s02	James	20.3	38

Reserves

sid	bid	day
s01	b01	2003
s01	b02	2004
s02	b01	2002
s02	b02	2003
s02	b03	2004

Boats

bid	bname	color
b01	Dragon	red
b02	Ocean	blue
b03	Roto	green

```
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))
```

- All boats {b01, b02, b03}
- When sid=s01, then all the boats reserved by s01: {b01, b02}
- When sid=s02, then all the boats reserved by s01: {b01, b02, b03}

Division in SQL

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

- An Alternative way to write the previous query without using EXCEPT

```
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))
```

- Intuitively, for each sailor we check that there is no boat that has not been reserved by this sailor.

Outline

- Introduction
- **Data Manipulation Language (DML)**
  - Simple SQL Queries
  - Nested SQL Queries
  - **Aggregation and Grouping**
  - Others, More Example: Relational Algebra and SQL
- Advanced Concepts (DDL)
  - Introduction to View
  - Introduction to Assertion/Trigger
  - Introduction to Database Programming

Aggregate Operators

- SQL allows the use of arithmetic expressions.
- SQL supports five aggregate operations, which can be applied on any column of a relation.

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)	The maximum value in the A column.
MIN (A)	The minimum value in the A column.

Examples

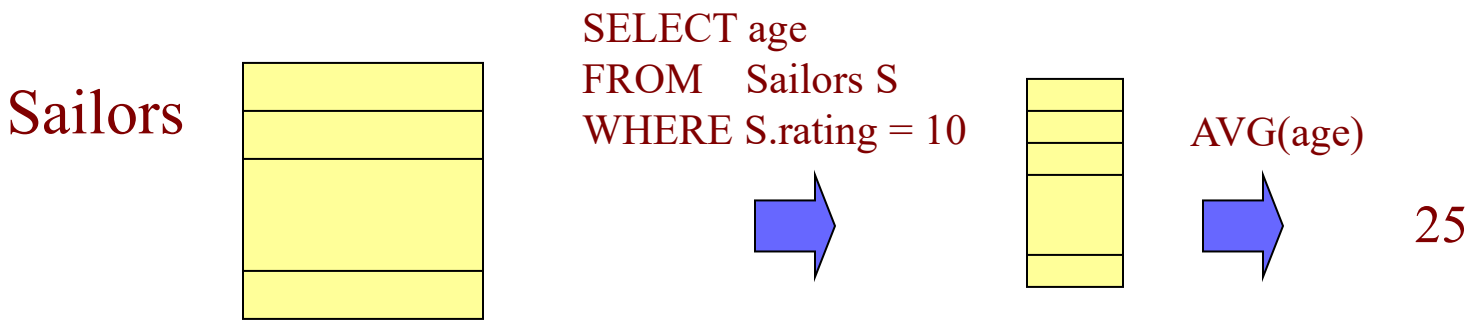
sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

- Q14: Find the average age of all sailors.

```
SELECT  AVG (S.age)
FROM    Sailors S
```

- Q15: Find the average age of sailors with rating of 10

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



Examples

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

- Q16: Find the name and age of the oldest sailor

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

If the SELECT clause uses an aggregate operation, then it must use only aggregate operations unless the query contains a GROUP BY clause.

```
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
```

Equivalent to the second query, and is allowed in the SQL/92 standard, but is **NOT** supported in some systems.

Examples

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

- Q17: Count the number of Sailors.

```
SELECT  COUNT (*)
FROM   Sailors S
```

- Q18: Count the number of different sailor names.

```
SELECT  COUNT (DISTINCT S.name)
FROM   Sailors S
```

Examples

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

- Aggregate operations offer an alternative to the ANY and ALL constructs.
- Q19: Find the names of sailors who are older than the oldest sailor with a rating of 10.

```
SELECT  S.name
FROM   Sailors S
WHERE  S.age > ( SELECT MAX (S2.age)
                 FROM  Sailors S2
                 WHERE  S2.rating = 10)
```

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.
- Q20: Find the age of the youngest sailor for each rating level.
  - Suppose we know that rating values go from 1 to 10; we can write 10 queries that look like this (!):

For i = 1, 2, ... , 10:

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

  - In general, we may not know how many rating levels exist, and what the rating values for these levels are!

Group by and Having

- To write such queries, we need a major extension to the basic SQL query form, namely the **GROUP BY** clause.
- The extension also includes an optional **HAVING** clause that can be used to specify qualifications over groups.

- Q20: Find the age of the youngest sailor for each rating level.

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

Group by

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

- Q20: Find the age of the youngest sailor for each rating level.

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

- For each group of tuples with the same rating, apply aggregate function min to age

sid	sname	rating	age
102	John	10	18
217	Mary	9	20
201	Athena	9	20
215	Joey	10	18
222	Rachel	5	25

Sailors table

sid	sname	rating	age
102	John	10	18
215	Joey	10	18
217	Mary	9	20
201	Athena	9	20
222	Rachel	5	25

rating	Min(age)
10	18
9	20
5	25

- The general format of **GROUP BY** and **HAVING**

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

- The target-list contains
  - (i) attribute names
  - (ii) terms with aggregate operations (e.g., MIN (S.age)).
- The attribute list (i) ***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.)

Conceptual Evaluation

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

1) Compute the cross-product of *relation-list*
2) Discard records that fail *qualification*
3) Partition the remaining records according to the value of attributes in *grouping-list*
4) Apply the *group-qualification* to eliminate groups that are not qualified
5) Delete attributes that are not specified by *target-list*, and *execute* aggregate operators.

Conceptual Evaluation

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

- The *group-qualification* is applied to eliminate some groups.
- Expressions in *group-qualification* must have *a single value per group* !
- Because one answer tuple is generated per qualifying group.
- In effect, an attribute in *group-qualification* must be either an argument of an aggregation operator or appears in *grouping-list*.

Examples

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

- Q21: Find the age of the youngest sailor who are at least 18 years, for each rating with at least 2 such sailors.

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

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

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

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

rating	
7	35.0

Answer relation

- Q22: Find the average age of sailors for each rating level that has at least two sailors.

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

```
SELECT  S.rating, AVG(S.age) AS avgage
FROM  Sailor S
GROUP BY S.rating
HAVING COUNT (*) > 1
```

OR

```
SELECT  S.rating, AVG(S.age) AS avgage
FROM  Sailor S
GROUP BY S.rating
HAVING 1 < (SELECT COUNT (*)
            FROM Sailors S2
            WHERE S.rating = S2.rating)
```

sid	sname	rating	age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Instance S3 of Sailor

Answer

Rating	avgage
3	44.5
7	40.0
8	40.5
10	25.5

We can use S.rating inside the nested subquery in the HAVING because it has a single value for the current group of sailors

- Q23: Find the average age of sailors who are at least 18 years old for each rating level that has at least two sailors.

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

sid	sname	rating	age
22	Dustin	7	44
25	Brutus	1	25
30	Lubber	7	22
22	Rusty	2	30
20	Zorba	7	17
24	Bouda	2	16

rating	age
7	44
7	22
1	25
2	30

S2

rating	age
7	44
7	22
7	17
1	25
2	30
2	16

Etc.

Result

rating	avgage
7	33
2	30

Examples

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

- Q24: For each red boat, find the number of reservations for this boat.

```
SELECT  B.bid,  COUNT (*) AS reservationcount
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 appear as arguments to an aggregate operator in the HAVING clause.

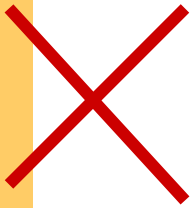
```
SELECT  B.bid,  COUNT (*) AS reservationcount
FROM  Boats B, Reserves R
WHERE  R.bid=B.bid AND B.color='red'
GROUP BY  B.bid
```

Examples

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

- Q25: Find those ratings for which the 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
                 GROUP BY S2.rating)
```



- ***Aggregate operations cannot be nested!***
  - This query will not work even if MIN(AVG(S2.age)), which is illegal, is allowed.
  - In the nested query, Sailors is partitioned into groups by rating, and the average age is computed for each rating value.
  - For each group, applying MIN to this average age value for the group will return the same value.

Examples

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

- Q25: Find those ratings for which the average age is the minimum over all ratings

```
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)
```

- It essentially computes a temporary table containing the average age for each rating value and
- then finds the rating(s) for which this average age is the minimum.

Summary of SQL Queries

- A query in SQL can consist of up to six clauses, but only the first two, SELECT and FROM, are mandatory. The clauses are specified in the following order:

SELECT <attribute list>
FROM <table list>
[**WHERE** <condition>]
[**GROUP BY** <grouping attribute(s)>]
[**HAVING** <group condition>]
[**ORDER BY** <attribute list>]

Outline

- Introduction
- **Data Manipulation Language (DML)**
  - Simple SQL Queries
  - Nested SQL Queries
  - Aggregation and Grouping
  - Others, More Example: Relational Algebra and SQL
- Advanced Concepts (DDL)
  - Introduction to View
  - Introduction to Assertion/Trigger

NULL Value

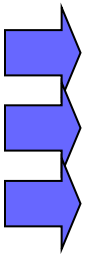
- Field value: UNKNOWN
  - For example: A new employee has not been assigned a supervisor yet.
- Field attribute inapplicable

Employee

eno	ename	supervisor_eno	spouse_name

eno	ename	supervisor_eno	spouse_name
123	Raymond	NULL	NULL
200	Peter	200	Mary
300	Mary	123	Peter

NULL Value



eno	ename	supervisor_eno		spouse_name
123	Raymond	NULL	UNKNOWN	NULL
200	Peter	300	FALSE	Mary
300	Mary	123	TRUE	Peter

```
SELECT *
FROM    employee
WHERE  supervisor_eno =  123
```

The condition following 'where' clause eliminates FALSE or unknown

eno	ename	supervisor_eno	spouse_name
300	Mary	123	Peter

Comparisons Involving NULL

- Meanings of NULL
  - **Unknown value**
  - **Unavailable or withheld value**
  - **Not applicable attribute**
- Each individual NULL value considered to be different from every other NULL value
- SQL uses a three-valued logic:
  - **TRUE**, **FALSE**, and **UNKNOWN**

Three-Valued Logic

Logical Connectives in Three-Valued Logic

(a)

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

(b)

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

(c)

NOT	
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

Comparisons Involving NULL

- Two rows are duplicates if matching columns are either equal or both NULL
  - NULL=NULL (for tuple)
  - (NULL=NULL) = Unknown (for comparison, i.e., where)
- NULL is counted in COUNT(*)
- All other aggregate discard NULL values

Outer Joins

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

List (sid, bid) for sailors and boats they have reserved.

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

sid	bid	day
22	101	10/10/05
58	103	11/12/05

SELECT R.sid, R.bid

FROM Sailors S NATURAL LEFT OUTER JOIN Reserves R

All left relation rows (Sailors) appear in the result

sid	bid
22	101
31	NULL
58	103

← Use of NULL

Outer Joins

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

List (sid, bid) for sailors and boats they have reserved.

sid	bid	day
22	101	10/10/05
58	103	11/12/05

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

SELECT R.sid, R.bid

FROM Reserves R NATURAL RIGHT OUTER JOIN Sailors S

sid	bid
22	101
31	NULL
58	103

All right relation rows (Sailors) appear in the result

Outer Joins

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

SB

sid	bid
22	101
31	NULL
58	103

bid	bname	color
101	apollo	white
103	maria	blue
107	princess	pink

SELECT SB.sid, B.bid

FROM SB NATURAL FULL OUTER JOIN Boats B

sid	bid
22	101
31	NULL
58	103
NULL	107

Example: Relational Algebra and SQL

- Exercise 1: 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)
- The key fields are underlined, and the domain of each field is listed after the field name. Therefore sid is the key for Suppliers, pid is the key for Parts, and sid and pid together form the key for Catalog.

Example

Suppliers	sid	sname	address
Parts	pid	pname	color
Catalog	sid	pid	cost

- Q1.1: Find the names of suppliers who supply some green part.

```
SELECT S.sname
FROM Suppliers S, Parts P, Catalog C
WHERE P.color='green' AND C.pid=P.pid AND C.sid=S.sid
```

$$\pi_{sname}(\pi_{sid}((\pi_{pid}\sigma_{color='green'}Parts)\bowtie Catalog)\bowtie Suppliers)$$

Example

Suppliers	sid	sname	address
Parts	pid	pname	color
Catalog	sid	pid	cost

- Q1.2: Find the sids of suppliers who supply some red or green part.

```
SELECT C.sid
FROM Catalog C, Parts P
WHERE ((P.color = 'red' OR P.color = 'green')
        AND P.pid = C.pid)
```

$\pi_{sid}(\pi_{pid}(\sigma_{color='red' \vee color='green'}Parts) \bowtie catalog)$

Example

Suppliers	sid	sname	address
Parts	pid	pname	color
Catalog	sid	pid	cost

- Q1.3: Find the sids of suppliers who supply every part.

(Solution 1)

```
SELECT C.sid
FROM Catalog C
WHERE NOT EXISTS (SELECT P.pid
                  FROM Parts P
                  EXCEPT
                  SELECT C1.pid
                  FROM Catalog C1
                  WHERE C1.sid = C.sid)
```

the pid of parts

the pid of the parts reserved by a particular supplier S_i

$(\pi_{sid,pid}Catalog)/(\pi_{pid}Parts)$

Example

Suppliers	sid	sname	address
Parts	pid	pname	color
Catalog	sid	pid	cost

- Q1.3: Find the sids of suppliers who supply every part.

  (Solution 2)

```
SELECT C.sid
FROM Catalog C
WHERE NOT EXISTS (SELECT P.pid
                  FROM Parts P
                  WHERE NOT EXISTS (SELECT *
                                    FROM Catalog C1
                                    WHERE C1.sid = C.sid
                                     AND C1.pid = P.pid))
```

$(\pi_{sid,pid}Catalog)/(\pi_{pid}Parts)$

Example

Suppliers	sid	sname	address
Parts	pid	pname	color
Catalog	sid	pid	cost

- Q1.4: Find the sids of suppliers who supply every red part or supply every green part. (Solution 1)

```
SELECT C.sid
FROM Catalog C
WHERE (NOT EXISTS (SELECT P.pid
                   FROM Parts P
                   WHERE P.color = 'red'
                   EXCEPT
                   SELECT C1.pid
                   FROM Catalog C1
                   WHERE C1.sid = C.sid))
   OR (NOT EXISTS (SELECT P1.pid
                   FROM Parts P1
                   WHERE P1.color = 'green'
                   EXCEPT
                   SELECT C2.pid
                   FROM Catalog C2
                   WHERE C2.sid = C.sid))
```

$\rho(R1, ((\pi_{sid,pid}\ Catalog)/(\pi_{pid}\sigma_{color='red'}\ Parts)))$
$\rho(R2, ((\pi_{sid,pid}\ Catalog)/(\pi_{pid}\sigma_{color='green'}\ Parts)))$
$R1 \cup R2$

Example

Suppliers	sid	sname	address
Parts	pid	pname	color
Catalog	sid	pid	cost

- Q1.4: Find the sids of suppliers who supply every red part or supply every green part. (Solution 2)

```
SELECT C.sid
FROM Catalog C
WHERE (NOT EXISTS (SELECT P.pid
                   FROM Parts P
                   WHERE P.color = 'red' AND
                     (NOT EXISTS (SELECT *
                                  FROM Catalog C1
                                  WHERE C1.sid = C.sid AND C1.pid = P.pid))))
    OR ( NOT EXISTS (SELECT P1.pid
                     FROM Parts P1
                     WHERE P1.color = 'green' AND
                       (NOT EXISTS (SELECT *
                                    FROM Catalog C2
                                    WHERE C2.sid = C.sid AND C2.pid = P1.pid))))
```

$\rho(R1, ((\pi_{sid,pid}\ Catalog)/(\pi_{pid}\sigma_{color='red'}\ Parts)))$
$\rho(R2, ((\pi_{sid,pid}\ Catalog)/(\pi_{pid}\sigma_{color='green'}\ Parts)))$
$R1 \cup R2$

Example

Suppliers	sid	sname	address
Parts	pid	pname	color
Catalog	sid	pid	cost

- Q1.5: Find pairs of different sids such that the supplier with the first sid charges more for some part than the supplier with the second sid (first sid != second sid).

```
SELECT C1.sid, C2.sid
FROM   Catalog C1, Catalog C2
WHERE C1.pid = C2.pid
    AND C1.sid != C2.sid AND C1.cost > C2.cost
```

ρ(R1,Catalog)

ρ(R2,Catalog)

$\pi_{R1.sid,R2.sid}(\sigma_{R1.pid=R2.pid \wedge R1.sid \neq R2.sid \wedge R1.cost>R2.cost}(R1 \times R2))$

Example

Suppliers	sid	sname	address
Parts	pid	pname	color
Catalog	sid	pid	cost

- Q1.6: Find the pids of the most expensive parts supplied by suppliers named Yosemite Sham

```
SELECT C.pid
FROM Catalog C, Suppliers S
WHERE C.sid = S.sid AND S.sname = 'Yosemite Sham' AND
        C.cost ≥ ALL (Select C2.cost
                      FROM Catalog C2, Suppliers S2
                      WHERE S2.sname = 'Yosemite Sham'
                      AND C2.sid = S2.sid)
```

$\rho(R1, \pi_{sid}\sigma_{sname='Yosemite\ Sham'}Suppliers)$
$\rho(R2, R1 \bowtie Catalog)$
$\rho(R3, R2)$
$\rho(R4(1 \rightarrow sid, 2 \rightarrow pid, 3 \rightarrow cost), \sigma_{R3.cost<R2.cost}(R3 \times R2))$
$\pi_{pid}(R2 - \pi_{sid,pid,cost}\ R4)$

Example

Suppliers	sid	sname	address
Parts	pid	pname	color
Catalog	sid	pid	cost

- Q1.6: Find the pids of parts supplied by every supplier at less than $200. (If any supplier either does not supply the part or charges more than $200 for it, the part is not selected.)

```
SELECT C.pid
FROM Catalog C
WHERE (NOT EXISTS (SELECT S.sid
                   FROM Suppliers S
                   EXCEPT
                   SELECT C1.sid
                   FROM Catalog C1
                   WHERE C1.pid = C.pid
                         AND C1.cost<= $200))
```

$(\pi_{sid,pid}\sigma_{cost<=\text{'\$200'}}\ Catalog)/(\pi_{sid}Suppliers)$

Example

Suppliers	sid	sname	address
Parts	pid	pname	color
Catalog	sid	pid	cost

- Q1.6: Find the pids of parts supplied by every supplier at less than $200. (If any supplier either does not supply the part or charges more than $200 for it, the part is not selected.)

```
SELECT C.pid
FROM Catalog C
WHERE (NOT EXISTS (SELECT S.sid
                   FROM Suppliers S
                   WHERE  (NOT EXISTS (SELECT *
                                       FROM Catalog C1
                                       WHERE C1.pid = C.pid
                                         AND C1.sid = S.sid
                                         AND C1.cost<= $200)))
```

$(\pi_{sid,pid}\sigma_{cost<='\$200'}\ Catalog)/(\pi_{sid}Suppliers)$

Example: Relational Algebra and SQL

- Exercise 2: Consider the following schema:
  - Flights(*flno:* integer, *from:* string, *to:* string, *distance:* integer, *departs:* time, *arrives:* time)
  - Aircraft(*aid:* integer, *aname:* string, *cruisingrange:* integer)
  - Certified(*eid:* integer, *aid:* integer)
  - Employees(*eid:* integer, *ename:* string, *salary:* integer)
- Note that the Employees relation describes pilots and other kinds of employees as well; every pilot is certified for some aircraft (otherwise, he or she would not qualify as a pilot), and only pilots are certified to fly.

Example

Flights	flno	from	to	distance	departs	arrive
Aircraft	aid	aname	cruisingrange			
Certified	eid	aid				
Employees	eid	ename	salary			

- Q2.1: Find the eids of pilots certified for some Boeing aircraft.

```
SELECT C.eid
FROM Aircraft A, Certified C
WHERE A.aid = C.aid AND A.aname = 'Boeing'
```

$$\pi_{eid}(\sigma_{aname='Boeing'}(Aircraft \bowtie Certified))$$

Example

Flights	flno	from	to	distance	departs	arrive
Aircraft	aid	aname	cruisingrange			
Certified	eid	aid				
Employees	eid	ename	salary			

- Q2.2: Find the names of pilots certified for some Boeing aircraft.

```
SELECT E.ename
FROM Aircraft A, Certified C, Employees E
WHERE A.aid = C.aid AND A.aname = 'Boeing' AND E.eid = C.eid
```

$\pi_{ename}(\sigma_{aname='Boeing'}(Aircraft \bowtie Certified \bowtie Employees))$

Example

Flights	flno	from	to	distance	departs	arrive
Aircraft	aid	aname	cruisingrange			
Certified	eid	aid				
Employees	eid	ename	salary			

- Q2.3: Find the aids of all aircraft that **CAN** be used on non-stop flights from Bonn to Madras (cruising range > distance).

```
SELECT A.aid
FROM Aircraft A, Flights F
WHERE F.from = 'Bonn' AND F.to = 'Madrid'
        AND A.cruisingrange > F.distance
```

$\rho(\text{BonnToMadrid}, \sigma_{\text{from='Bonn'} \wedge \text{to='Madrid'}}(\text{Flights}))$

$\pi_{\text{aid}}(\sigma_{\text{cruisingrange}>\text{distance}}(\text{Aircraft} \times \text{BonnToMadrid}))$

Example

Flights	flno	from	to	distance	departs	arrive
Aircraft	aid	aname	cruisingrange			
Certified	eid	aid				
Employees	eid	ename	salary			

- Q2.4: Identify the flights that can be piloted by every pilot whose salary is more than $100,000.

```
SELECT E.ename
FROM Aircraft A, Certified C, Employees E, Flights F
WHERE A.aid = C.aid AND E.eid = C.eid AND  salary > 100,000
    AND distance < cruisingrange
```

$$\pi_{flno}(\sigma_{distance<cruisingrange \wedge salary>100,000}(Flights \bowtie Aircraft \bowtie Certifed \bowtie Employees))$$

Example

Flights	flno	from	to	distance	departs	arrive
Aircraft	aid	aname	cruisingrange			
Certified	eid	aid				
Employees	eid	ename	salary			

- Q2.5: Find the names of pilots who can operate planes with a range greater than 3,000 miles but are not certified on any Boeing aircraft.

```
SELECT E.ename
FROM Certified C, Employees E, Aircraft A
WHERE A.aid = C.aid AND E.eid = C.eid AND A.cruisingrange > 3000
    AND E.eid NOT IN (
                        SELECT C2.eid
                        FROM    Certified C2, Aircraft A2
                        WHERE C2.aid = A2.aid AND A2.aname = 'Boeing' )
```

$\rho(R1, \pi_{eid}(\sigma_{cruisingrange>3000}(Aircraft \bowtie Certified)))$

$\pi_{ename}(Employees \bowtie (R1 - \pi_{eid}(\sigma_{aname='Boeing'}(Aircraft \bowtie Certified))))$

Example

Flights	flno	from	to	distance	departs	arrive
Aircraft	aid	aname	cruisingrange			
Certified	eid	aid				
Employees	eid	ename	salary			

- Q2.6: Find the eids of employees who make the highest salary.

```
SELECT E.eid
FROM Employees E
WHERE E.salary = ( Select MAX (E2.salary)
                   FROM Employees E2  )
```

ρ(E1,Employees)

ρ(E2,Employees)

ρ(E3, $\pi_{E2.eid}$(E1 $\bowtie_{E1.salary>E2.salary}$ E2)

(π_{eid}E1) − E3

Example

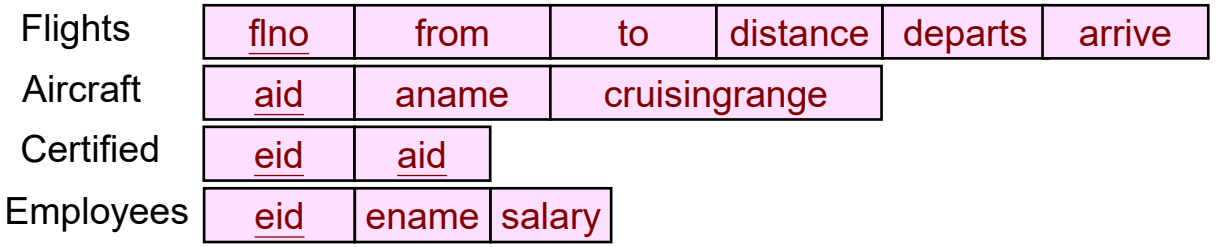
Flights	flno	from	to	distance	departs	arrive
Aircraft	aid	aname	cruisingrange			
Certified	eid	aid				
Employees	eid	ename	salary			

- Q2.7: Find the eids of employees who make the second highest salary.

```
SELECT E.eid
FROM Employees E
WHERE E.salary = (SELECT MAX (E2.salary)
                  FROM    Employees E2
                  WHERE E2.salary != (SELECT MAX (E3.salary)
                                      FROM    Employees E3 ))
```

- ρ(E1,Employees)
- ρ(E2,Employees)
- ρ(E3, $\pi_{E2.eid}$(E1 $\bowtie_{E1.salary>E2.salary}$ E2)
- ρ(E4,E2 ⋈ E3)
- ρ(E5,E2 ⋈ E3)
- ρ(E6, $\pi_{E5.eid}$(E4 $\bowtie_{E1.salary>E5.salary}$ E5)
- (π_{eid}E3) − E6

Example

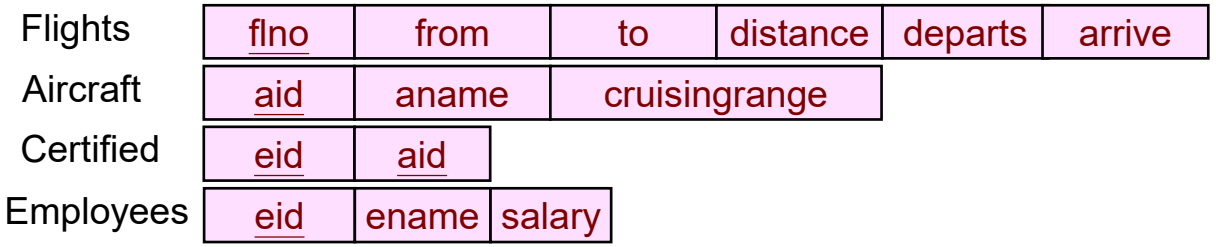


- Q2.8: Find the eids of employees who are certified for the largest number of aircraft.

```
SELECT Temp.eid
FROM ( SELECT C.eid AS eid, COUNT (C.aid) AS cnt,
          FROM Certified C
          GROUP BY C.eid) AS Temp
WHERE Temp.cnt = ( SELECT MAX (Temp.cnt)
                   FROM Temp)
```

This cannot be expressed in basic relational algebra

Example



- Q2.9: Find the eids of employees who are certified for exactly three aircraft.

```
SELECT C1.eid
FROM Certified C1, Certified C2, Certified C3
WHERE (C1.eid = C2.eid AND C2.eid = C3.eid AND
        C1.aid != C2.aid AND C2.aid != C3.aid AND C3.aid != C1.aid)
EXCEPT
SELECT C4.eid
FROM Certified C4, Certified C5, Certified C6, Certified C7,
WHERE (C4.eid = C5.eid AND C5.eid = C6.eid AND C6.eid = C7.eid
        AND C4.aid != C5.aid AND C4.aid != C6.aid
        AND C4.aid != C7.aid AND C5.aid != C6.aid
        AND C5.aid != C7.aid AND C6.aid != C7.aid )
```

Example

Flights	flno	from	to	distance	departs	arrive
Aircraft	aid	aname	cruisingrange			
Certified	eid	aid				
Employees	eid	ename	salary			

- Q2.9: Find the eids of employees who are certified for exactly three aircraft.

$\rho(R1, Certified)$

$\rho(R2, Certified)$

$\rho(R3, Certified)$

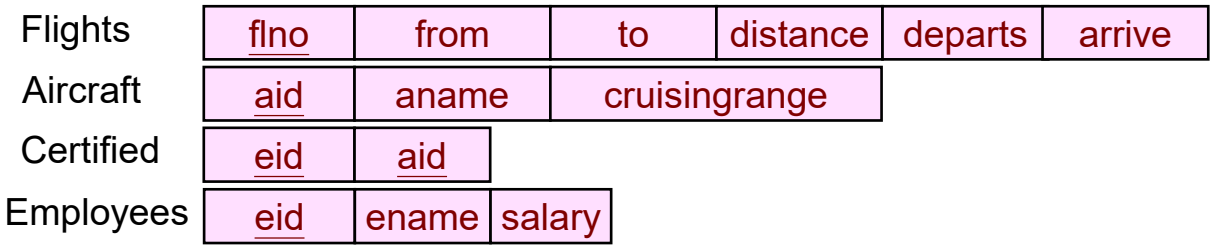
$\rho(R4, Certified)$

$\rho(R5, \pi_{eid}(\sigma_{(R1.eid=R2.eid=R3.eid) \wedge (R1.aid \; != \; R2.aid \; != \; R3.aid)}(R1 \times R2 \times R3)))$

$\rho(R6, \pi_{eid}(\sigma_{(R1.eid=R2.eid=R3.eid=R4.eid) \wedge (R1.aid \; != R2.aid \; != R3.aid \; != R4.aid)}(R1 \times R2 \times R3 \times R4)))$

$R5 - R6$

Example

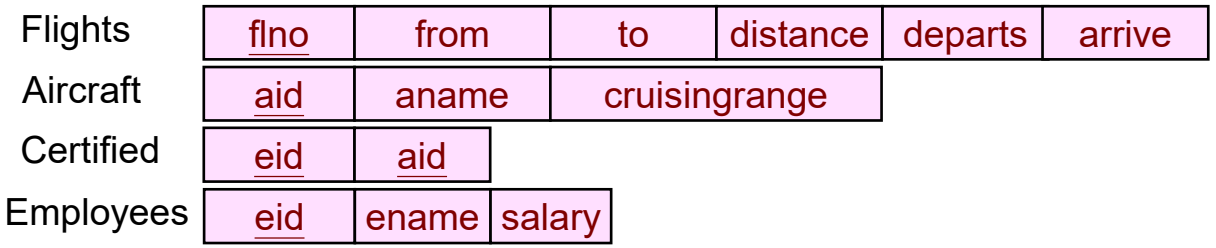


- Q2.10: Find the total amount paid to employees as salaries

```
SELECT SUM (E.salary)
FROM Employees E
```

This cannot be expressed in basic relational algebra

Example



- Q2.11: Is there a sequence of flights from Madison to Timbuktu? Each flight in the sequence is required to depart from the city that is the destination of the previous flight; the first flight must leave Madison, the last flight must reach Timbuktu, and there is no restriction on the number of intermediate flights. Your query must determine whether a sequence of flights from Madison to Timbuktu exists for any input Flights relation instance.

- *This cannot be expressed in relational algebra and SQL.*

Outline

- Introduction
- Data Manipulation Language (DML)
  - Simple SQL Queries
  - Nested SQL Queries
  - Aggregation and Grouping
  - Others, More Example: Relational Algebra and SQL
- **Advanced Concepts (DDL)**
  - **Introduction to View**
  - Introduction to Assertion/Trigger

Views in SQL

- A view is a “virtual” table that is derived from other tables
- Allows for limited update operations
- Since the table may not physically be stored
- Allows full query operations
- A convenience for expressing certain operations
- Provide a mechanism to hide certain data from the view of certain users

Views

List (sid, bid) for sailors and boats they have reserved.

```
CREATE VIEW  SB (sid, bid)
    AS SELECT R.sid, R.bid
        FROM Reserves R NATURAL RIGHT OUTER JOIN Sailors S
```

sid	bid
22	101
31	NULL
58	103

View SB is a table which is not explicitly stored in the database.

A view can be used just like a base table.

Using a Virtual Table

- We can specify SQL queries on a newly create table

  (view):

```
SELECT COUNT(*)
FROM    SB
```

- When no longer needed, a view can be dropped:

  DROP SB;

Efficient View Implementation

- Query modification approach:
  - Present the view query in terms of a query on the underlying base tables
- Disadvantage:
  - Inefficient for views defined via complex queries
  - Especially if additional queries are to be applied to the view within a short time period

Efficient View Implementation

- View materialization approach
  - Physically create a temporary view table when the view is first queried
  - Keep that table on the assumption that other queries on the view will follow
  - Requires efficient strategy for automatically updating the view table when the base tables are updated
- Incremental update strategies
  - DBMS determines what new tuples must be inserted, deleted, or modified in a materialized view table

Update Views

- Update on a single view without aggregate operations:
  - Update may map to an update on the underlying base table
- Views involving joins:
  - An update may map to an update on the underlying base relations
  - Not always possible

Outline

- Introduction
- Data Manipulation Language (DML)
  - Simple SQL Queries
  - Aggregation/Grouping
  - Nested SQL Queries
  - More Example: Relational Algebra and SQL
- **Advanced Concepts (DDL)**
  - Introduction to View
  - **Introduction to Assertion/Trigger**

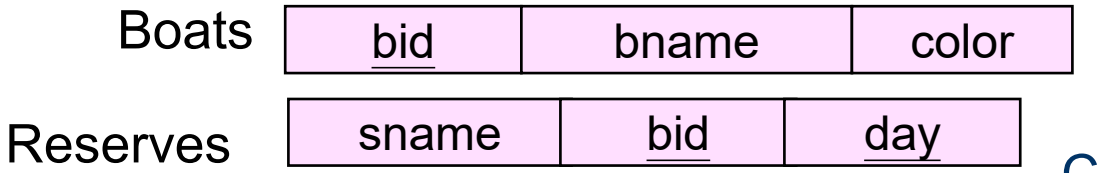
General Constraints

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

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

Sailors

sid	sname	rating	age



Constraints can be named.

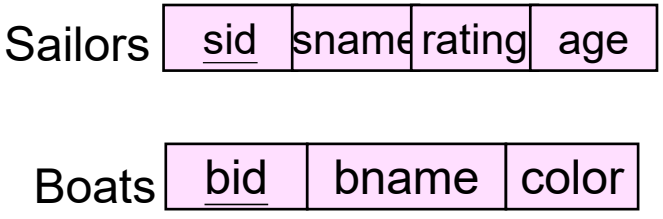
Constraint:

Interlake boats cannot be reserved.

```
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))
)
```

When a boat is inserted into Reserves or an existing row is modified, the conditional expression in the CHECK constraint is evaluated. If it evaluates to false, the command is rejected.

Constraints Over Multiple Relations



Constraint:
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!
- A *constraint is not required to hold for an empty relation*

Specifying Constraints as Assertions

- CREATE ASSERTION
  - Specify additional types of constraints outside scope of built-in relational model constraints
- 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
)
```

Sailors

sid	sname	rating	age

Boats

bid	bname	color

SQL Triggers

- Objective: to monitor a database and take initiate action when a condition occurs
- Triggers are expressed in a syntax similar to assertions and include the following:
  - Event
    - Such as an insert, deleted, or update operation
  - Condition
  - Action
    - To be taken when the condition is satisfied

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

Action

Sailors

sid	sname	rating	age

Outline

- Introduction
- Data Manipulation Language (DML)
  - Simple SQL Queries
  - Aggregation/Grouping
  - Nested SQL Queries
  - More Example: Relational Algebra and SQL
- **Advanced Concepts (DDL)**
  - Introduction to View
  - Introduction to Assertion/Trigger

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.
- 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.