SQL: The Query Language

Guifeng Zheng
School of Software
SUN YAT-SEN UNIVERSITY

The important thing is not to stop questioning.
Albert Einstein

Agenda

- What’s SQL
- RA to SQL
- Simple SQL Query
- Advanced SQL Query
- Constraints & Access Control
- DB Programing

Review

- Relational Algebra (Operational Semantics操作语义)
  - Given a query, how to mix and match the relational algebra operators to answer it
  - Used for query optimization用于查询优化
- Relational Calculus (Declarative Semantics说明性语义)
  - Given a query, what do I want my answer set to include?
- Algebra and safe calculus are simple and powerful models for query languages for relational model
  - Have same expressive power有相同的表达力
- SQL can express every query that is expressible in relational algebra/calculus. (and more)

Next topic: SQL

- Standard language for querying and manipulating data

Structured Query Language

- Many standards: ANSI SQL, SQL92/SQL2, SQL3/SQL99
- Originally: Structured English Query Language (SEQUEL)
- Vendors support various subsets/extensions
- We'll do Oracle/MySQL/generic
  - "No one ever got fired for buying Oracle."

The SQL Query Language

- *非过程化：只提出"做什么"*
- *独立：可独立用于联机交互*
- *嵌入式：可嵌入到高级语言中*

- Basic form (many more bells and whistles in addition):

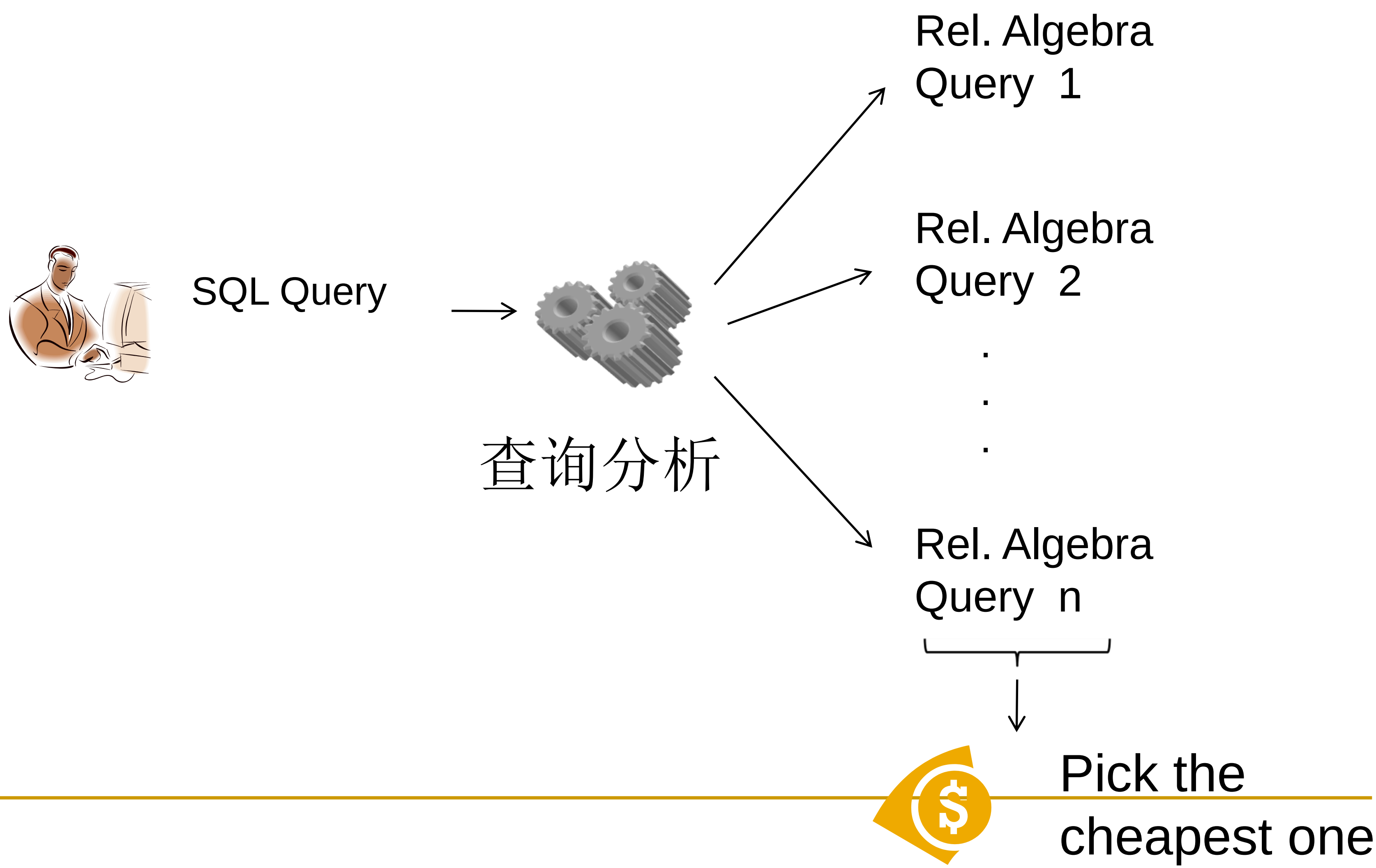
SELECT **attributes**
FROM **relations (possibly multiple, joined)**
WHERE **conditions (selections)**

Relational Query Languages

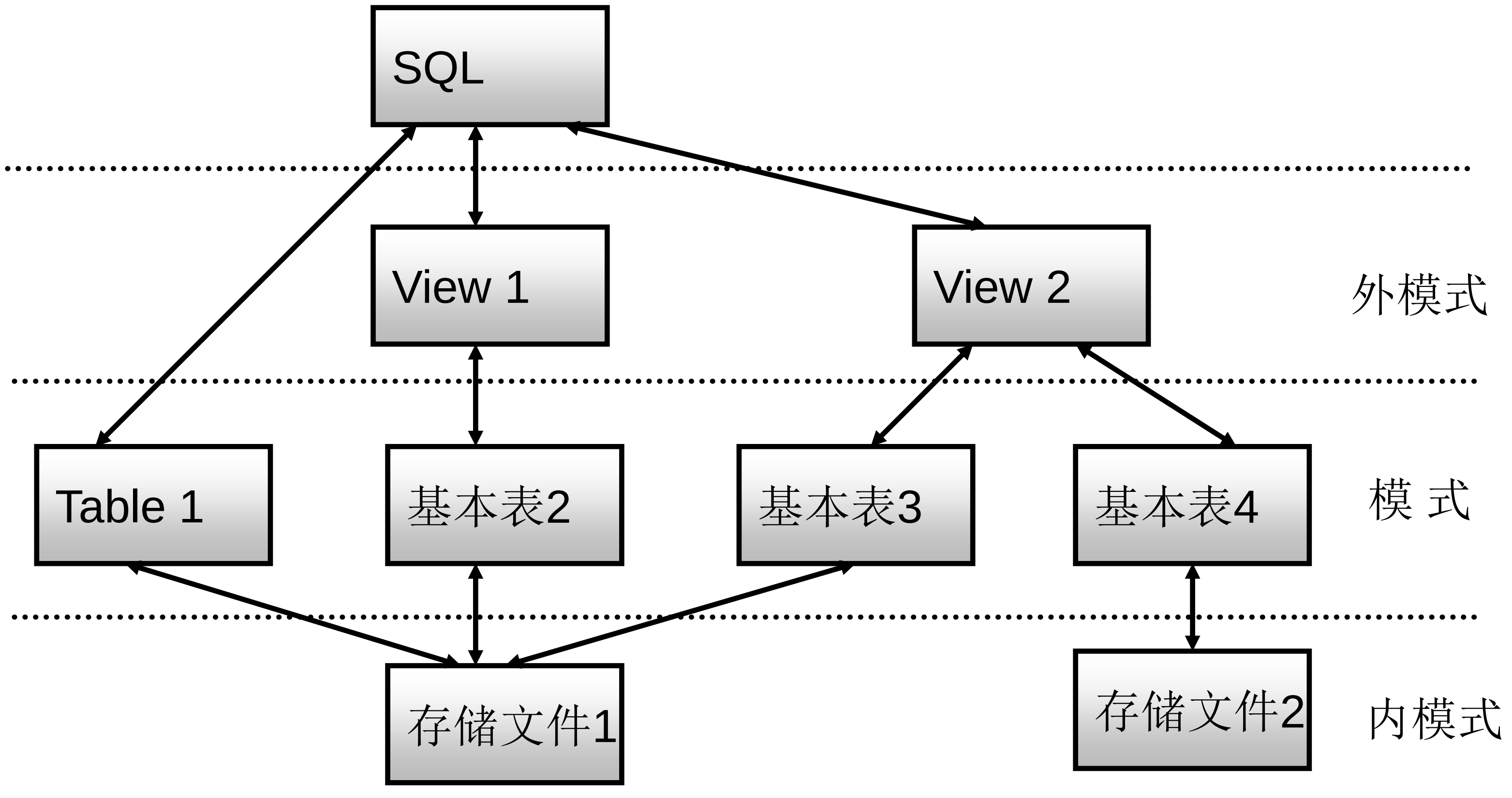
- Two sublanguages:
  - DDL – Data Definition定义 Language
    - Define and modify schema (at all 3 levels)
  - DML – Data Manipulation操作 Language
    - Queries can be written intuitively.

- DBMS is responsible for efficient evaluation.
  - The key: precise semantics for relational queries.
  - Optimizer can re-order operations
    - Won’t affect query answer.
  - Choices driven by “cost model” 成本模型

Relational Query Languages



Big Picture



SQL Clauses

SQL 语言的动词

SQL 功 能	动 词
数 据 查 询	SELECT
数 据 定 义	CREATE，DROP，ALTER
数 据 操 纵	INSERT，UPDATE DELETE
数 据 控 制	GRANT，REVOKE

Basic Data Types in SQL

- Characters:
  - CHAR(20) -- fixed length
  - VARCHAR(40) -- variable length
- Numbers:
  - BIGINT, INT, SMALLINT, TINYINT
  - REAL, FLOAT -- differ in precision
  - MONEY
- Times and dates:
  - DATE
  - DATETIME -- SQL Server

RA → SQL

$$\Pi_L(\sigma_C(R_1 \times \ldots R_n)$$

```
SELECT L
FROM   R1, ..., Rn
WHERE  C
```

RA → SQL

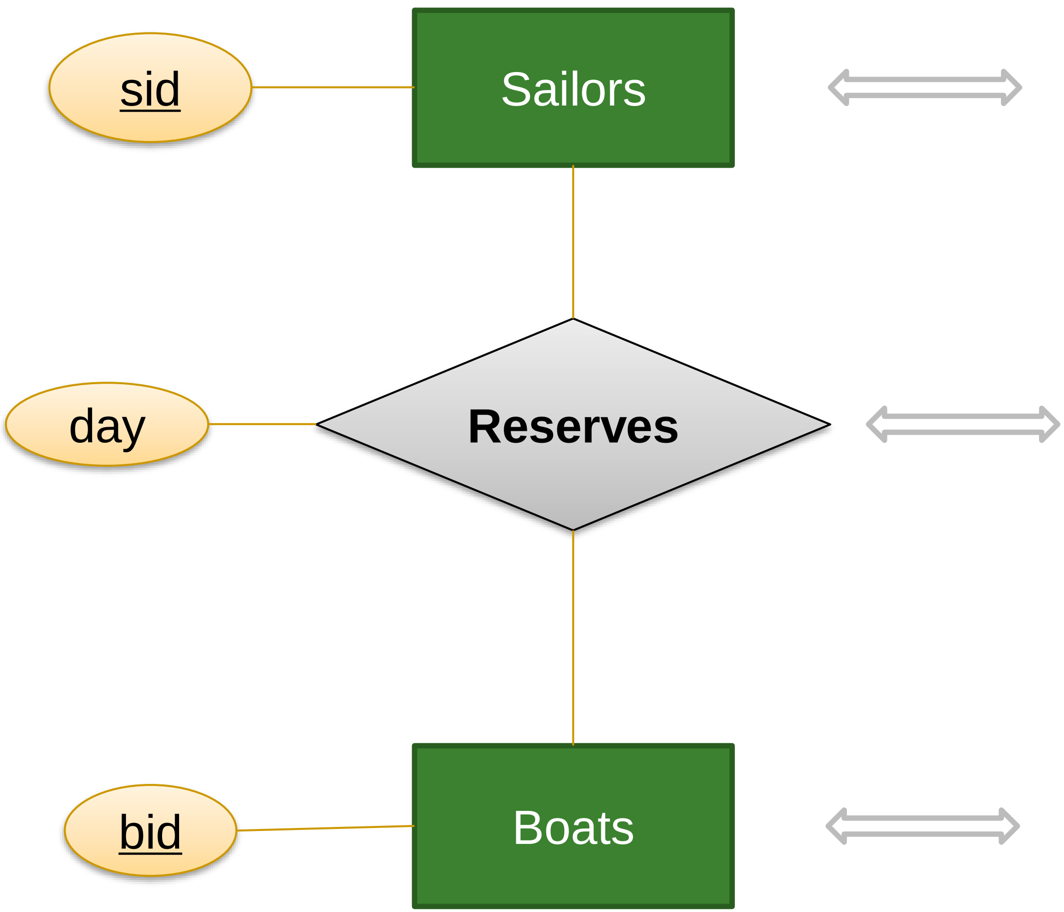
$\Pi_L(\sigma_C(R_1 \times \ldots R_n))$

- SQL SELECT → RA Projection Π
- SQL WHERE → RA Selection σ
- SQL FROM → RA Join/cross
  - Comma-separated list…
- SQL renaming → RA rho ρ
- More ops later

```
SELECT  L
FROM    R1, ..., Rn
WHERE   C
```

- *Keep RA in the back of your mind…*

Example Database



<u>sid</u>	sname	rating	age
1	Fred	7	22
2	Jim	2	39
3	Nancy	8	27

FOREIGN KEY 外键

<u>sid</u>	<u>bid</u>	<u>day</u>
1	102	9/12
2	102	9/13

<u>bid</u>	bname	color
101	Nina	red
102	Pinta	blue
103	Santa Maria	red

The SQL DDL

```
CREATE TABLE Sailors (
   sid INTEGER,
   sname CHAR(20),
   rating INTEGER,
   age REAL,
   PRIMARY KEY sid);

 CREATE TABLE Reserves (
   sid INTEGER,
   bid INTEGER,
   day DATE,
  PRIMARY KEY (sid, bid, day),
  FOREIGN KEY sid REFERENCES Sailors,
  FOREIGN KEY bid REFERENCES Boats);


CREATE TABLE Boats (
   bid INTEGER,
   bname CHAR (20),
   color CHAR(10)
   PRIMARY KEY bid);
```

sid	sname	rating	age
1	Fred	7	22
2	Jim	2	39
3	Nancy	8	27

FOREIGN KEY 外键

sid	**bid**	**day**
1	102	9/12
2	102	9/13

bid	bname	color
101	Nina	red
102	Pinta	blue
103	Santa Maria	red

The SQL DML

Sailors

sid	sname	rating	age
1	Fred	7	22
2	Jim	2	39
3	Nancy	8	27

- Find all 18-year-old sailors:

$$\sigma_{age=18}(Sailors)$$

```
SELECT *
FROM  Sailors S
WHERE S.age=18
```

- To find just names and ratings, replace the first line:

$$\pi_{sname,rating}(\sigma_{age=18}(Sailors))$$

```
SELECT S.sname,S.rating
```

Basic SQL Query

DISTINCT: optional. Answer should not contain duplicates.

SQL default: duplicates are *not* eliminated! (Result a “multiset”)

target-list : List of expressions over attributes of tables in *relation-list*

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

qualification : Comparisons combined using AND, OR and NOT. Comparisons are Attr *op* const or Attr1 *op* Attr2, where *op* is one of $=, <, >, \neq$, etc.

relation-list : List of relation names, possibly with a *range-variable* after each name

Query Semantics

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

1. FROM : compute *cross product* of tables.
2. WHERE : Check conditions, discard tuples that fail.
3. SELECT : Delete unwanted fields.
4. DISTINCT *(optional)* : eliminate duplicate rows.

Note: Probably the least efficient way to compute a query!

- *Query optimizer* will find more efficient ways to get the *same answer*.

SQL Query Semantics

```
SELECT a1, a2, …, ak
FROM   R1 AS x1, R2 AS x2, …, Rn AS xn
WHERE  Conditions
```

Parallel assignment – all tuples

```
Answer = {}
for all assignments x1 in R1, …, xn in Rn do
   if Conditions then
      Answer = Answer ∪ {(a1,…,ak)}
return Answer
```

Doesn't impose any order

SQL Query Semantics

```
SELECT a1, a2, …, ak
FROM   R1 AS x1, R2 AS x2, …, Rn AS xn
WHERE  Conditions
```

Nested loops:

```
Answer = {}
for x1 in R1 do
   for x2 in R2 do
      …..
         for xn in Rn do
            if Conditions then
               Answer = Answer ∪ {(a1,…,ak)}
return Answer
```

Advanced SQL Query

- Querying Multiple Relations
- Self-Join
- Arithmetic Expressions
- String Comparisons
- Set-Comparison
- Nested Queries
- Correlation Queries

Querying Multiple Relations

Cross Product

```
SELECT S.sname
FROM   Sailors S, Reserves R
WHERE  S.sid=R.sid AND R.bid=102
```

Natural Join

Sailors

sid	sname	rating	age
1	Fred	7	22
2	Jim	2	39
3	Nancy	8	27

Reserves

sid	bid	day
1	102	9/12
2	102	9/13

Find sailors who've reserved at least one boat

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

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

About Range Variables

- Needed when ambiguity could arise.
  - e.g., same table used multiple times in FROM ("self-join")

```
SELECT  x.sname, x.age, y.sname, y.age
FROM    Sailors x, Sailors y
WHERE   x.age > y.age
```

Sailors x

sid	sname	rating	age
1	Fred	7	22
2	Jim	2	39
3	Nancy	8	27

Sailors y

sid	sname	rating	age
1	Fred	7	22
2	Jim	2	39
3	Nancy	8	27

Arithmetic Expressions 算术表达式

```
SELECT S.age, S.age-5 AS age1, 2*S.age AS age2
FROM   Sailors S
WHERE  S.sname = ‘dustin’
```

```
SELECT S1.sname AS name1, S2.sname AS name2
FROM   Sailors S1, Sailors S2
WHERE  2*S1.rating = S2.rating - 1
```

String Comparisons字符串比较

```
SELECT S.sname
FROM   Sailors S
WHERE  S.sname LIKE ‘B_%B’
```

BoB
BaoB
BaoooB
…

- `_’ stands for any one character and
- `%’ stands for 0 or more arbitrary characters.

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

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

... or:

```
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'
```

Find sid’s of sailors who’ve reserved a red **and** a green boat

```
SELECT R.sid
FROM   Boats B,Reserves R
WHERE  R.bid=B.bid AND
  (B.color=‘red’ AND B.color=‘green’)
```

Find sid's of sailors who've reserved 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'
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'
```

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

- Could use a self-join:

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

Find sid's of sailors who have not reserved a boat

```
SELECT S.sid
FROM   Sailors S

EXCEPT

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

Nested Queries: IN

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)
```

Nested Queries: NOT IN

Names of sailors who've ***not*** *reserved boat #103:*

```
SELECT  S.sname
FROM    Sailors S
WHERE   S.sid NOT IN
    (SELECT  R.sid
     FROM    Reserves R
     WHERE  R.bid=103)
```

Nested Queries with Correlation

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)
```

- Subquery must be recomputed for each Sailors tuple.
  - Think of subquery as a function call that runs a query

More on Set-Comparison Operators

- we've seen: IN, EXISTS
- can also have: NOT IN, NOT EXISTS
- other forms: *op* ANY, *op* ALL
- 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')
```

Next: Grouping & Aggregation

- In SQL:
  - aggregation operators in SELECT,
  - Grouping in GROUP BY clause

- Recall aggregation operators:
  - sum, avg, min, max, count
    - strings, numbers, dates
  - Each applies to scalars
  - Count also applies to row: count(*)
  - Can DISTINCT inside aggregation op: count(DISTINCT x)

- Grouping: group rows that agree on single value
  - Each group becomes one row in result

Aggregation functions

- Numerical: SUM, AVG, MIN, MAX
- Char: MIN, MAX
  - In lexocographic/alphabetic order
- Any attribute: COUNT
  - Number of values

- SUM(B) = 10
- AVG(A) = 1.5
- MIN(A) = 1
- MAX(A) = 3
- COUNT(A) = 4

A	B
1	2
3	4
1	2
1	2

Straight aggregation

- In R.A. $\Pi_{\text{sum(x)}\rightarrow\text{total}}(R)$
- In SQL:

```
SELECT SUM(x) total
FROM   R
```

- Just put the aggregation op in SELECT
- NB: aggreg. ops applied to each non-null val
  - count(x) counts the number of nun-null vals in field x
  - Use count(*) to count the number of rows

Straight aggregation example

- COUNT applies to duplicates, unless otherwise stated:

```
SELECT Count(category)
FROM   Product
WHERE  year > 1995
```

same as Count(*), except excludes nulls

- Better:

```
SELECT COUNT(DISTINCT category)
FROM   Product
WHERE  year > 1995
```

- Can we say:

```
SELECT category, COUNT(category)
FROM   Product
WHERE  year > 1995
```

Straight aggregation example

- Purchase(product, date, price, quantity)

- Q: Find total sales for the entire database:

```
SELECT SUM(price * quantity)
FROM   Purchase
```

- Q: Find total sales of bagels:

```
SELECT SUM(price * quantity)
FROM   Purchase
WHERE  product = 'bagel'
```

Largest balance again

- Acc(name,bal,type)
- Q: Who has the largest balance?
- Q: Who has the largest balance of each type?

- Can we do these with aggregation functions?

Straight grouping

- Group rows together by field values
- Produces one row for each group
  - I.e., by each (combin. of) grouped val(s)
  - Don't select non-grouped fields

```
SELECT   product
FROM     Purchase
GROUP BY product
```

- Reduces to DISTINCT selections:

```
SELECT DISTINCT product
FROM   Purchase
```

Grouping & aggregation

- Sometimes want to group and compute aggregations *by group*
  - Aggregation op applied to rows in group,
  - not to all rows in table

- Purchase(product, date, price, quantity)
- Find total sales for products that sold for > 0.50:

```
SELECT   product, SUM(price*quantity) total
FROM     Purchase
WHERE    price > .50
GROUP BY product
```

Illustrated G&A example

Purchase

Product	Date	Price	Quantity
Bagel	10/21	0.85	15
Banana	10/22	0.52	7
Banana	10/19	0.52	17
Bagel	10/20	0.85	20

Illustrated G&A example

- First compute the FROM-WHERE
- Then GROUP BY product:

Product	Date	Price	Quantity
Banana	10/19	0.52	17
Banana	10/22	0.52	7
Bagel	10/20	0.85	20
Bagel	10/21	0.85	15

Illustrated G&A example

- Finally, aggregate and select:

Product	TotalSales
Bagel	$29.75
Banana	$12.48

```
SELECT   product, SUM(price*quantity) total
FROM     Purchase
WHERW    price > .50
GROUP BY product
```

Illustrated G&A example

- GROUP BY may be reduced to (a possibly more complicated) subquery:

```
SELECT   product, SUM(price*quantity) total
FROM     Purchase
WHERE    price > .50
GROUP BY product
```

```
SELECT DISTINCT x.product, (SELECT SUM(y.price*y.quantity)
                             FROM   Purchase y
                             WHERE  x.product = y.product
                              AND y.price > .50) total
FROM   Purchase x
WHERE  x.price > .50
```

Multiple aggregations

Product	SumSales	MaxQuantity
Banana	$12.48	17
Bagel	$29.75	20

For every product, what is the total sales and max quantity sold?

```
SELECT   product, SUM(price * quantity) SumSales,
                  MAX(quantity) MaxQuantity
FROM     Purchase
WHERE    price > .50
GROUP BY product
```

Another grouping/aggregation e.g.

- Movie(title, year, length, studioName)
- Q: How many total minutes of film have been produced by *each* studio?
- Strategy: Divide movies into groups per studio, then add lengths *per group*

Another grouping/aggregation e.g.

```
SELECT   studio, sum(length) totalLength
FROM     Movies
GROUP BY studio
```

Title	Year	Length	Studio
Star Wars	1977	120	Fox
Jedi	1980	105	Fox
Aviator	2004	800	Miramax
Pulp Fiction	1995	110	Miramax
Lost in Translation	2003	95	Universal

Another grouping/aggregation e.g.

```
SELECT    studio, sum(length) length
FROM      Movies
GROUP BY studio
```

Title	Year	Length	Studio
Star Wars	1977	120	Fox
Jedi	1980	105	Fox
Aviator	2004	800	Miramax
Pulp Fiction	1995	110	Miramax
Lost in Translation	2003	95	Universal

Another grouping/aggregation e.g.

```
SELECT    studio, sum(length) totalLength
FROM      Movies
GROUP BY studio
```

Title	Year	Length	Studio
Star Wars	1977	120	Fox
Jedi	1980	105	Fox
Aviator	2004	800	Miramax
Pulp Fiction	1995	110	Miramax
Lost in Translation	2003	95	Universal

→

Studio	Length
Fox	225
Miramax	910
Universal	95

Grouping/aggregation example

- StarsIn(SName,Title,Year)
- Q: Find the year of each star's first movie

```
SELECT   sname, min(year) firstyear
FROM     StarsIn
GROUP BY sname
```

- Q: Find the span of each star's career
  - Look up first and last movies

Account types again

- Acc(name,bal,type)
- Q: Who has the largest balance *of each type*?

- Can we do this with grouping/aggregation?

G & A for constructed relations

- Movie(title,year,producerSsn,length)
- MovieExec(name,ssn,netWorth)

- Can do the same thing for larger, non-atomic relations
- Q: How many mins. of film did each producer make?
  - What happens to non-producer movie-execs?

```
SELECT   name, sum(length) total
FROM     Movie, MovieExec
WHERE    producerSsn = ssn
GROUP BY name
```

HAVING clauses

- Sometimes want to limit which rows may be grouped
- Q: How many mins. of film did each rich producer make?
  - Rich = netWorth > 10000000

```
SELECT    name, sum(length) total
FROM      Movie, MovieExec
WHERE     producerSsn = ssn
GROUP BY  name
HAVING    netWorth > 10000000
```

- Q: Is HAVING necessary here?
- A: No, could just add rich req. to WHERE

HAVING clauses

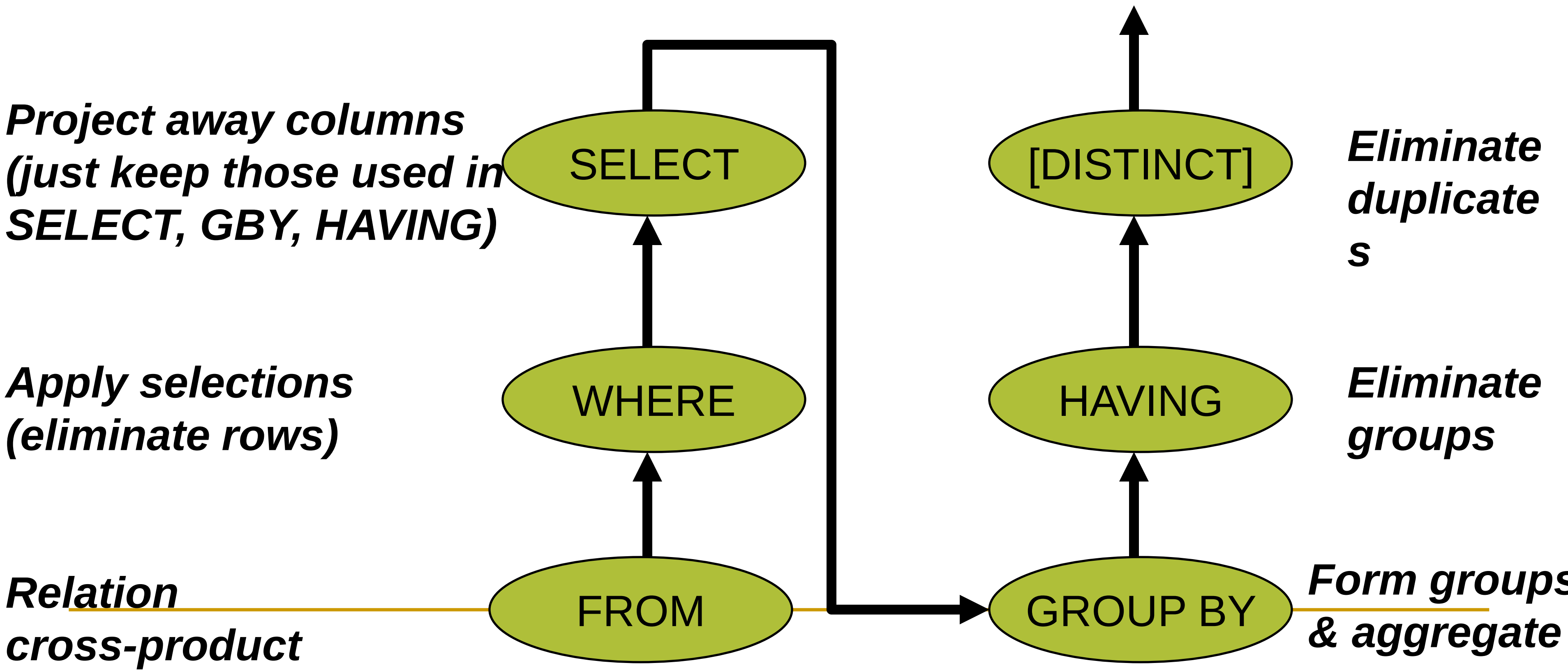
- Sometimes want to limit which rows may be grouped
- Q: How many mins. of film did each rich producer make?
  - Old = made movies before 1930

```
SELECT    name, sum(length) total
FROM      Movie, MovieExec
WHERE     producerSsn = ssn
GROUP BY name
HAVING    min(year) < 1930
```

- Q: Is HAVING necessary here?

Conceptual SQL Evaluation

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



Sorting the Results of a Query

- ORDER BY *column* [ASC | DESC] [, ...]

```
SELECT  S.rating, S.sname, S.age
      FROM  Sailors S, Boats B, Reserves R
      WHERE  S.sid=R.sid
            AND R.bid=B.bid AND B.color='red'
      ORDER BY  S.rating, S.sname;
```

- Can order by any column in SELECT list, including expressions or aggs:

```
SELECT  S.sid, COUNT (*) AS redrescnt
      FROM  Sailors S, Boats B, Reserves R
      WHERE  S.sid=R.sid
            AND R.bid=B.bid AND B.color='red'
      GROUP BY S.sid
      ORDER BY  redrescnt DESC;
```

New topic: Nulls in SQL

- If we don’t have a value, can put a NULL

- Null can mean several things:
  - Value does not exists
  - Value exists but is unknown
  - Value not applicable

- But null is not the same as 0
  - See Douglas Foster Wallace…

Null Values

- x = NULL → 4*(3-x)/7 = NULL
- x = NULL → x + 3 – x = NULL
- x = NULL → 3 + (x-x) = NULL
- x = NULL → x = 'Joe' is UNKNOWN

- In general: no row using null fields appear in the selection test will pass the test
  - With one exception
- *Pace* Boole, SQL has three boolean values:
  - FALSE = 0
  - TRUE = 1
  - UNKNOWN = 0.5

Null values in boolean expressions

- C1 AND C2 = min(C1, C2)
- C1 OR C2 = max(C1, C2)
- NOT C1 = 1 – C1

```
SELECT *
FROM   Person
WHERE  (age < 25) AND
       (height > 6 OR weight > 190)
```

E.g.
age=20
height=NULL
weight=180

- height > 6 = UNKNOWN
- → UNKNOWN OR weight > 190 = UNKOWN
- → (age < 25) AND UNKNOWN = UNKNOWN

Comparing null and non-nulls

- The schema specifies whether null is allowed for each attribute
  - NOT NULL to forbid
  - Nulls are allowed by default

- Unexpected behavior:

```
SELECT *
FROM   Person
WHERE  age < 25  OR  age >= 25
```

- Some Persons are not included!
- The “trichotomy law” does not hold!

Testing for null values

- Can test for NULL explicitly:
  - x IS NULL
  - x IS NOT NULL
- But:
  - x = NULL *is never true*

```
SELECT *
FROM   Person
WHERE  age < 25  OR  age >= 25 OR age IS NULL
```

- Now it includes all Persons

Null/logic review

- TRUE AND UNKNOWN = ?
- TRUE OR UNKNOWN = ?
- UNKNOWN OR UNKNOWN = ?
- X = NULL = ?
- http://en.wikipedia.org/wiki/Null_(SQL)

Joins

```
SELECT (column_list)
FROM  table_name
  [INNER | {LEFT |RIGHT | FULL } OUTER] JOIN table_name
    ON qualification_list
WHERE …
```

Explicit join semantics needed

unless it is an INNER join (INNER is default)

Inner Join

Only rows that match the qualification are returned.

```
SELECT s.sid, s.name, r.bid
  FROM Sailors s INNER JOIN Reserves r
                ON s.sid = r.sid
```

Returns only those sailors who have reserved boats.

SELECT s.sid, s.name, r.bid
FROM Sailors s INNER JOIN Reserves r
ON s.sid = r.sid

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
95	Bob	3	63.5

sid	bid	day
22	101	10/10/96
95	103	11/12/96

s.sid	s.name	r.bid
22	Dustin	101
95	Bob	103

Left Outer Join

- Returns all matched rows
- plus all unmatched rows from the table on the left of the join clause

(use nulls in fields of non-matching tuples)

```
SELECT s.sid, s.name, r.bid
FROM Sailors s LEFT OUTER JOIN Reserves r
                    ON s.sid = r.sid
```

```
SELECT s.sid, s.name, r.bid
FROM Sailors s LEFT OUTER JOIN Reserves r
ON s.sid = r.sid
```

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
95	Bob	3	63.5

sid	bid	day
22	101	10/10/96
95	103	11/12/96

s.sid	s.name	r.bid
22	Dustin	101
95	Bob	103
31	Lubber	

Right Outer Join

Right Outer Join returns all matched rows, plus all unmatched rows from the table on the right of the join clause

```
SELECT r.sid, b.bid, b.name
FROM Reserves r RIGHT OUTER JOIN Boats b
                ON r.bid = b.bid
```

SELECT r.sid, b.bid, b.name
FROM Reserves r RIGHT OUTER JOIN Boats b
ON r.bid = b.bid

sid	bid	day
22	101	10/10/96
95	103	11/12/96

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

r.sid	b.bid	b.name
22	101	Interlake
	102	Interlake
95	103	Clipper
	104	Marine

Full Outer Join

Full Outer Join returns all (matched or unmatched) rows from the tables on both sides of the join clause

```
SELECT r.sid, b.bid, b.name
FROM Reserves r FULL OUTER JOIN Boats b
ON r.bid = b.bid
```

SELECT r.sid, b.bid, b.name
FROM Reserves r FULL OUTER JOIN Boats b
ON r.bid = b.bid

sid	bid	day
22	101	10/10/96
95	103	11/12/96

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

r.sid	b.bid	b.name
22	101	Interlake
	102	Interlake
95	103	Clipper
	104	Marine

Note: in this case it is the same as the ROJ!
bid is a foreign key in reserves, so all reservations must have a corresponding tuple in boats.

Views: Defining External DB Schemas

```
CREATE VIEW view_name
AS select_statement
```

Makes development simpler

Often used for security

Not “materialized”

```
CREATE VIEW Reds
AS SELECT  B.bid,  COUNT (*) AS scount
     FROM Boats B, Reserves R
     WHERE  R.bid=B.bid AND   B.color=‘red’
     GROUP BY  B.bid
```

Views Instead of Relations in Queries

```
CREATE VIEW Reds
AS SELECT  B.bid,  COUNT (*) AS scount
     FROM Boats B, Reserves R
     WHERE  R.bid=B.bid AND   B.color=‘red’
     GROUP BY  B.bid
```

bid	scount
102	1

Reds

```
SELECT  bname, scount
    FROM Reds R, Boats B
    WHERE  R.bid=B.bid
         AND scount < 10
```

Discretionary Access Control

GRANT *privileges* ON *object* TO *users* [WITH GRANT OPTION]

- **Object can be a Table or a View**
- **Privileges can be:**
  - **Select**
  - **Insert**
  - **Delete**
  - **References (cols) – allow to create a foreign key that references the specified column(s)**
  - **All**
- **Can later be REVOKEd**
- **Users can be single users or groups**
- **See Chapter 17 for more details.**

Two more important topics

- Constraints
- SQL embedded in other languages

Integrity Constraints (Review)

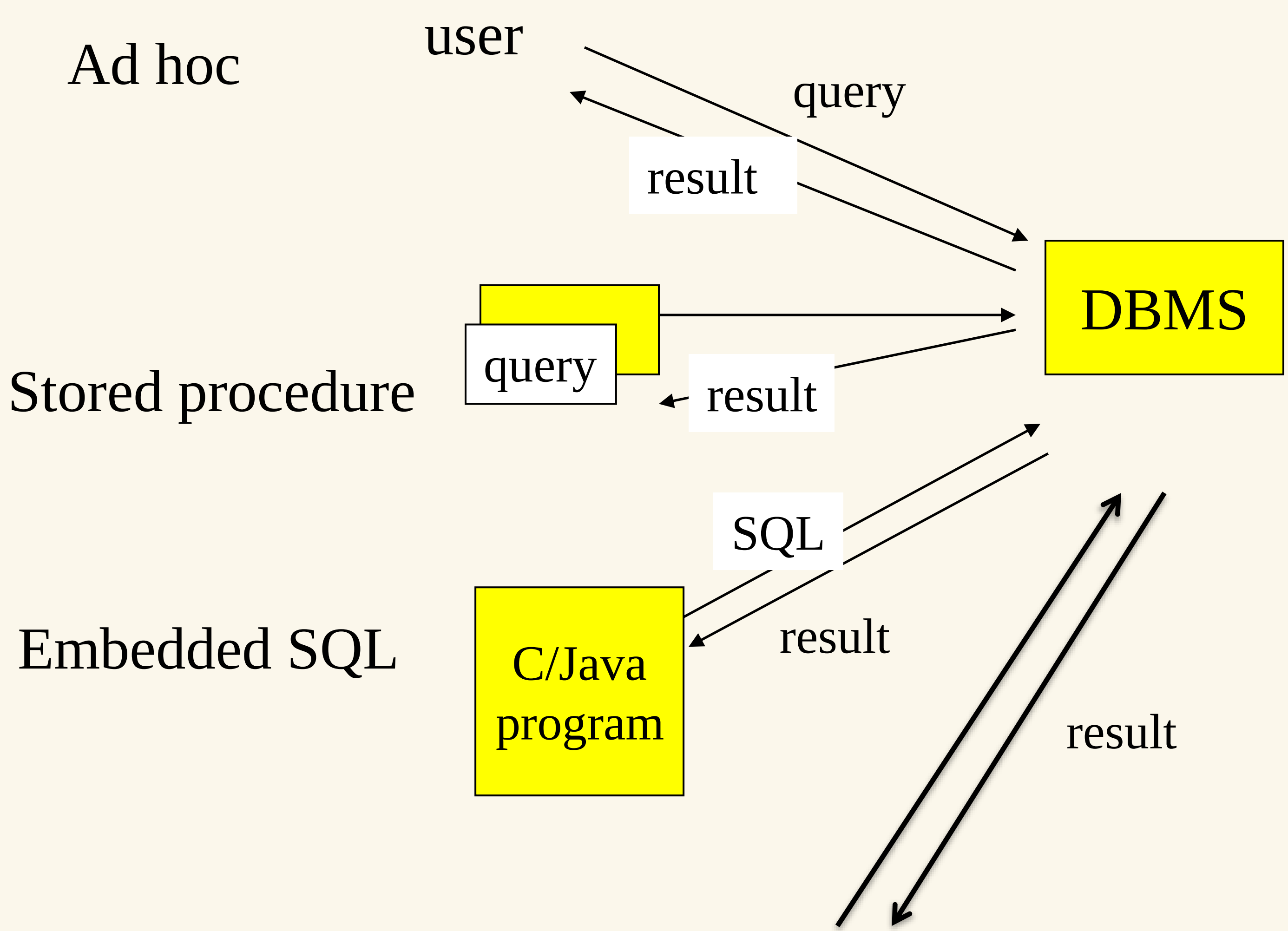
- An IC describes conditions that every *legal instance* of a relation must satisfy.
  - Inserts/deletes/updates that violate IC's are disallowed.
  - Can ensure application semantics (e.g., *sid* is a key), or prevent inconsistencies (e.g., *sname* has to be a string, *age* must be < 200)
- *Types of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

General Constraints

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

```
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)))
```



Database API: ODBC, JDBC, PHP

Writing Applications with SQL

- SQL is not a general purpose programming language.
  - \+ Tailored for data retrieval and manipulation
  - \+ Relatively easy to optimize and parallelize
  - \- Can't write entire apps in SQL alone

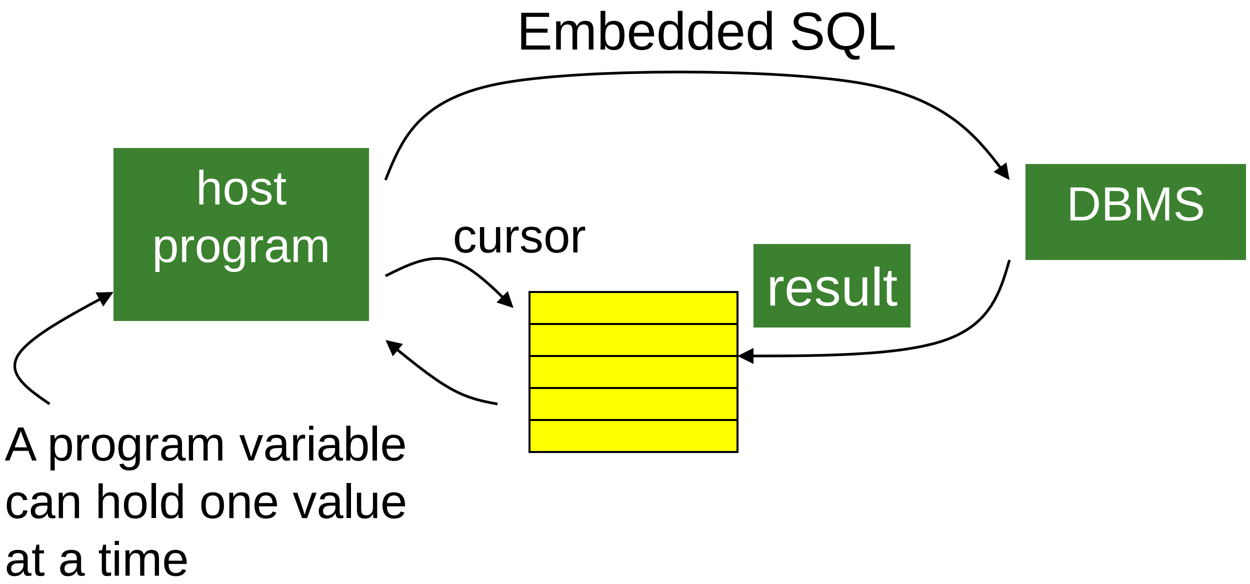
Options:

- Make the query language "Turing complete"
  - Avoids the "impedance mismatch"
  - but, loses advantages of relational language simplicity
- Allow SQL to be embedded in regular programming languages.
- Q: What needs to be solved to make the latter approach work?

Embedded SQL

- DBMS vendors traditionally provided "host language bindings"
  - E.g. for C or COBOL
  - Allow SQL statements to be called from within a program
  - Typically you preprocess your programs
  - Preprocessor generates calls to a proprietary DB connectivity library
- General pattern
  - One call to *connect* to the right database (login, etc.)
  - SQL statements can refer to host variables from the language
- Typically vendor-specific
  - We won't look at any in detail, we'll look at standard stuff
- Problem
  - SQL relations are (multi-)sets, no *a priori* bound on the number of records. No such data structure in C.
  - SQL supports a mechanism called a *cursor* to handle this.

Why is cursor needed?



Cursor bridges the gap between value-oriented host program and set-oriented DBMS

Example Embedded SQL

From within a host language, find the names and account numbers of customers with more than the variable *amount* dollars in some account.

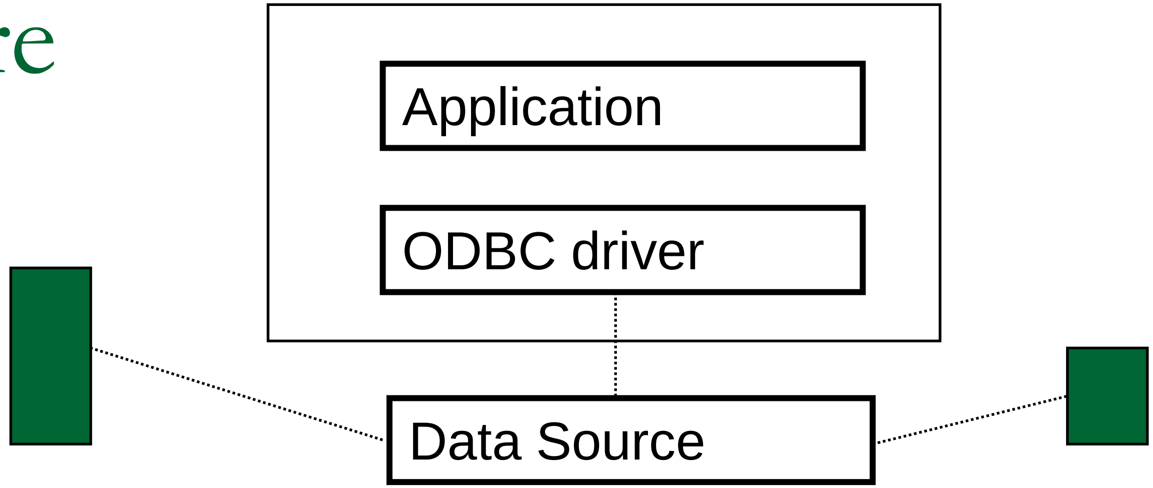
- Specify the query in SQL and declare a *cursor* for it

```
EXEC SQL
      declare c cursor for
      select customer-name, account-number
      from depositor, account
      where depositor.account-number = account.account-number and
                  account.balance > :amount
END-EXEC
```

Database APIs: Alternative to embedding

- Rather than modify compiler, add a library with database calls (API)
  - special objects/methods
  - passes SQL strings from language, presents result sets in a language-friendly way
  - *ODBC* a C/C++ standard started on Windows
  - *JDBC* a Java equivalent
  - Most scripting languages have similar things
    - E.g. For Perl there is DBI, “oraPerl”, other packages
- Mostly DBMS-neutral
  - at least try to hide distinctions across different DBMSs

Architecture



- A lookup service maps “data source names” (“DSNs”) to drivers
  - Typically handled by OS
- Based on the DSN used, a “driver” is linked into the app at runtime
- The driver traps calls, translates them into DBMS-specific code
- Database can be across a network
- ODBC is standard, so the same program can be used (in principle) to access multiple database systems
- Data source may not even be an SQL database!

ODBC/JDBC

- Various vendors provide drivers
  - MS bundles a bunch into Windows
  - Vendors like DataDirect and OpenLink sell drivers for multiple OSes
- Drivers for various data sources
  - Relational DBMSs (Oracle, DB2, SQL Server, etc.)
  - “Desktop” DBMSs (Access, Dbase, Paradox, FoxPro, etc.)
  - Spreadsheets (MS Excel, Lotus 1-2-3, etc.)
  - Delimited text files (.CSV, .TXT, etc.)
- You can use JDBC/ODBC *client*s over many data sources
  - E.g. MS Query comes with many versions of MS Office (msqry32.exe)
- Can write your own Java or C++ programs against xDBC

JDBC

- Part of Java, easy to use
- Java comes with a JDBC-to-ODBC bridge
  - So JDBC code can talk to any ODBC data source
  - E.g. look in your Windows Control Panel or MacOS Utilities folder for JDBC/ODBC drivers!
- JDBC tutorial online
  - http://developer.java.sun.com/developer/Books/JDBCTutorial/

Next: Dynamic Web page

- http://pages.stern.nyu.edu/~mjohnson/dbms/php/hello.php

```
<html>
<head><title>Hello from PHP</title>
</head>
<body>
Here comes the PHP part:<BR><BR>
<?php print "Hello, World!<br>\n"; ?>
<br>That's it!
</body></html>
```

- Q: What the difference between
 and \n?

PHP vars

- Names always start with $
  - http://pages.stern.nyu.edu/~mjohnson/dbms/php/math.php

```
<?
   $num1 = 58;
   $num2 = 67;
   print "First number " . $num1 . "<br>";
   print "Second number " . $num2 . "<br>";
   $total = $num1 + $num2;
   print "The sum is " . $total . "<br>";
?>
```

Combining PHP and HTML

- http://pages.stern.nyu.edu/~mjohnson/dbms/php/combine.php

```
<?php
  for($z=0;$z<=5;$z++) {
?>
      Iteration number <? = $z ?><br>
<?
  }
?>
```

PHP & MySQL

1. Open a connection and open our DB:

```
$db = mysql_connect(“localhost",      user, pass);
mysql_select_db("test", $db);
```

2. Run query:

```
$result = mysql_query($query,$db);
```

PHP & MySQL

3. Extract next row of data from the results:

```
$myrow = mysql_fetch_row($result)
```

- What this means: myrow is an array that can then be accessed
- Other options, see code

- In general, to scroll through results, do:

```
while ($myrow = mysql_fetch_row($result))
        # print row’s data
```

API Summary

APIs are needed to interface DBMSs to programming languages

- Embedded SQL uses “native drivers” and is usually faster but less standard
- ODBC (used to be Microsoft-specific) for C/C++
- JDBC the standard for Java
- Scripting languages (PHP, Perl, JSP) are becoming the preferred technique for web-based systems

Summary

- Relational model has well-defined query semantics
- SQL provides functionality close to basic relational model

  (some differences in duplicate handling, null values, set operators, …)
- Typically, many ways to write a query
  - DBMS figures out a fast way to execute a query, regardless of how it is written.

Review

- Examples from sqlzoo.net

```
SELECT L
FROM   R1, …, Rn
WHERE  C
```

$$\Pi_L(\sigma_C(R_1 \times \ldots R_n))$$