SQL SELECT

Principle Form of a Query

```
SELECT desired attributes
FROM list of relations
WHERE qualification
(where clause is optional)

Example:
SELECT sname, rating
FROM Skaters
WHERE rating > 9 OR age < 12
```

sid	sname	rating	age
28	yuppy	9	15
31	debby	7	10
22	conny	5	10
58	lilly	10	13

sname	rating
debby	7
conny	5
lilly	10

Principle Form of a Query

```
SELECT rating,age
FROM Skaters
WHERE rating >= 10 OR age > 15
```

- Conversion to Relational Algebra
  - $\pi_{\texttt{rating,age}} (\sigma_{\texttt{rating>=10} \vee \texttt{age>15}} (\texttt{Skaters}))$
  - Start with the relation in the FROM clause
  - Apply σ, using condition in WHERE clause (selection)
  - Apply ∏, using attributes in SELECT clause (projection)
- Operational Semantics as in Relational Algebra
  - Imagine a tuple variable ranging over all tuples of the relation
  - For each tuple: check if is satisfies the **WHERE** clause. If so, print the attributes in **SELECT**.

sid	sname	rating	age
28	yuppy	9	15
31	debby	7	10
22	conny	5	10
58	lilly	10	13

sname	rating
debby	7
conny	5
lilly	10

Set vs. Multi-Set

- Difference SQL and RELATIONAL ALGEBRA
  - No elimination of duplicates (as long as no violation of primary key / unique constraint)
  - Tables in relational databases are generally NO sets (but "multi-sets")
  - Results of SQL queries are generally NO sets

```
SELECT age
FROM Skaters
```

<u>sid</u>	sname	rating	age
28	yuppy	9	15
31	debby	7	10
22	conny	5	10
58	lilly	10	13

age
15
10
10
13

Selection: The WHERE Clause

- Comparison terms:
  - **attr1 op const**:
    - age > 10
  - **attr1 op attr2**:
    - age < rating
  - **op** is one of <**,** =**,** >**,** <>**,!=,** <=**,** >=, **LIKE**
  - We may apply the usual arithmetic operations +, *, etc. to numeric values before we compare
    - Example: rating more than double the age
    - WHERE rating > 2*age

Selection: The WHERE Clause

- Boolean Operators:
  - Comparisons combined using **AND, OR** and **NOT**
    - **name =‘Cheng’ AND NOT age = 18**
- Strings
  - **name LIKE ‘%e_g’** (%: any string, _:any character)
  - Further string operations, e.g., concatenation, string-length, etc.
  - show all names that end in “y”
    - name LIKE ‘%y’
  - show all names that have an “i” in the second position
    - name LIKE ‘_i%’

Projection: Attribute Lists

- **Distinct**
  - Duplicate elimination

```
SELECT DISTINCT age
FROM Skaters
```

<u>sid</u>	sname	rating	age
28	yuppy	9	15
31	debby	7	10
22	conny	5	10
58	lilly	10	13

age
15
10
13

Projection: Attribute Lists

- *<u>Star</u>* as list of all attributes
  - show all skaters with a rating smaller than 9

```
SELECT *
 FROM Skaters
WHERE rating < 9
```

Good coding practice is to actually list the column names you NEED for the application functionality and not use * .

<u>sid</u>	sname	rating	age
28	yuppy	9	15
31	debby	7	10
22	conny	5	10
58	lilly	10	13

<u>sid</u>	sname	rating	age
31	debby	7	10
22	conny	5	10

Attribute Lists

- *Renaming; Expressions and constants* as values in columns

```
SELECT sname, rating AS reality,
              rating+1 AS upgrade,
              10 AS dream
FROM Skaters
```

The AS is not really needed

sid	sname	rating	age
28	yuppy	9	15
31	debby	7	10
22	conny	5	10
58	lilly	10	13

sname	reality	upgrade	dream
yuppy	9	10	10
debby	7	8	10
conny	5	6	10
lilly	10	11	10

Attribute Lists

- *Ordered Output*
  - ascending first by age then rating

```
SELECT *
FROM Skaters
ORDER BY age,rating
```

sid	sname	rating	age
28	yuppy	9	15
31	debby	7	10
22	conny	5	10
58	lilly	10	13

sid	sname	rating	age
22	conny	5	10
31	debby	7	10
58	lilly	10	13
28	yuppy	9	15

Multirelational Queries: Cross-Product

- List of relations in **`FROM`** clause

```
Skaters  X Participates:

SELECT *

FROM Skaters, Participates
```

Multirelational Queries: Join

- equals cross-product and selection
- Have to indicate comparison even with natural join
- Relation-dot-attribute disambiguates attributes from several relations.

– Example: "give me the names of all skaters that participate in a competition

```
SELECT sname
FROM Skaters, Participates
WHERE Skaters.sid = Participates.sid

SELECT sname
FROM Skaters JOIN Participates
ON Skaters.sid = Participates.sid
```

$$\pi_{\mathtt{sname}} (\mathtt{Participates} \bowtie \mathtt{Skaters})$$

Multirelational Queries: Join

```
SELECT sname
FROM Skaters, Participates
WHERE Skaters.sid = Participates.sid
```

$\pi_{\texttt{sname}}$ $(\texttt{Skaters} \bowtie \texttt{Participates})$

sid	sname	rating	age
28	yuppy	9	15
31	debby	7	10
22	conny	5	10

sid	cid	rank
31	101	2
22	103	7
31	103	1

sname
debby
debby
conny

Range Variables

- Optional use of range variables

```
SELECT S.sname
FROM Skaters S, Participates P
WHERE S.sid = P.sid AND P.cid = 101
```

- Use of range variable required when the same relation appears twice in the FROM clause
- Example: "*find pairs of skaters that have participated in the same competition*"

```
SELECT p1.sid, p2.sid
FROM Participates p1, Participates p2
WHERE p1.cid = p2.cid AND p1.sid < p2.sid
```

(note that r1.sid < r2.sid is needed to avoid producing (22,22) and to avoid producing a pair in both directions.)

Union, Intersection, Difference

- UNION, INTERSECT, EXCEPT
- Input relations for set operators must be set-compatible, I.e. they must have
  - Same number of attributes
  - The attributes, taken in order, must have same type
- As default, result relation is a set!!! (no multiset)
- Many systems do not provide primitives for intersection and difference

Union

- Skaters(sid,sname,rating,age)

  Participates(sid,cid,rank)

  Competition(cid,date,type)
- *Find skaters' sid that have participated in a regional or a local competition*

```
SELECT P.sid
FROM Participates P, Competition C
WHERE P.cid = C.cid AND
       (C.type = 'regional' OR C.type = 'local')

SELECT P.sid
FROM Participates P, Competition C
WHERE P.cid = C.cid AND C.type = 'local'
UNION
SELECT P.sid
FROM Participates P, Competition C
WHERE P.cid = C.cid AND C.type = 'regional'
```

Difference???

Using Join

```
SELECT P.sid
FROM Participates P, Competition C
WHERE P.cid = C.cid AND
           (C.type = 'regional' OR C.type = 'local')
```

C

cid	date	type
101	12/13/2014	local
103	01/12/2015	regional
104	01/20/2015	national

P

sid	cid	rank
31	101	2
58	103	7
58	101	7

sid
31
58
58

Using Union

C

cid	date	type
101	12/13/2014	local
103	01/12/2015	regional
104	01/20/2015	national

P

sid	cid	rank
31	101	2
58	103	7
58	101	7

```
SELECT P.sid
FROM Participates P, Competition C
WHERE P.cid = C.cid AND C.type =‘local’
UNION
SELECT P.sid
FROM Participates P, Competition C
WHERE P.cid = C.cid AND C.type =‘regional’
```

sid
31
58

Union

sid
58

=

sid
31
58

Intersection

- *Find skaters' sid that have participated in a regional and a local competition*

(1)
```
SELECT P.sid
FROM Participates P, Competition C
WHERE P.cid = C.cid AND C.type = 'local'
INTERSECT
SELECT P.sid
FROM Participates P, Competition C
WHERE P.cid = C.cid AND C.type = 'regional'
```

Join instead of Intersection

- *Find skaters' sid that have participated in a regional and a local competition*

(2)
```
SELECT P1.sid
FROM Participates P1, Participates P2, Competition C1,
Competition C2
WHERE (P1.cid = C1.cid AND C1.type = 'local') AND
      (P2.cid = C2.cid AND C2.type = 'regional') AND
       P1.sid = P2.sid)
```

cid	date	type
101	12/13/2014	local
103	01/12/2015	regional
104	01/20/2015	national

sid	cid	rank
31	101	2
58	103	7
58	101	7

sid	cid	rank
31	101	2
58	103	7
58	101	7

cid	date	type
101	12/13/2014	local
103	01/12/2015	regional
104	01/20/2015	national

Cross-Product after first two selections

Cross-Product after first two selections

```
SELECT P1.sid
FROM Participates P1, Competition C1,
WHERE (P1.cid = C1.cid AND C1.type = ‘local’)
```

C1

cid	date	type
101	12/13/2014	local
103	01/12/2015	regional
104	01/20/2015	national

P1

sid	cid	rank
31	101	2
58	103	7
58	101	7

sid
31
58

Cross-Product after first two selections

sid
31
58

Cross-Product after first two selections

```
SELECT P1.sid
FROM Participates P2, Competition C2,
WHERE (P2.cid = C2.cid AND C2.type = 'regional')
```

sid
31
58

C2

<u>cid</u>	date	type
101	12/13/2014	local
103	01/12/2015	regional
104	01/20/2015	national

P2

<u>sid</u>	<u>cid</u>	rank
31	101	2
58	103	7
58	101	7
C		

Cross-Product after first two selections

```
SELECT P1.sid
FROM Participates P2, Competition C2,
WHERE (P2.cid = C2.cid AND C2.type = 'regional')
```

sid
31
58

C2

cid	date	type
101	12/13/2014	local
103	01/12/2015	regional
104	01/20/2015	national

P2

sid	cid	rank
31	101	2
58	103	7
58	101	7

sid
58

Cross-Product after first two selections

```
SELECT P1.sid
FROM Participates P2, Competition C2,
WHERE (P2.cid = C2.cid AND C2.type = 'regional')
```

sid
31
58

C2

cid	date	type
101	12/13/2014	local
103	01/12/2015	regional
104	01/20/2015	national

P2

sid	cid	rank
31	101	2
58	103	7
58	101	7
C		

sid
58

Cross-Product after first two selections

sid
31
58

sid
58

Cross-Product after first two selections

```
SELECT P1.sid
FROM Participates P1, Competition C1, Participates P2,
Competition C2
WHERE (P1.cid = C1.cid AND C1.type = ‘local’) AND
      (P2.cid = C2.cid and C2.type = ‘regional’) AND
      (P1.sid = P2.sid)
```

sid
31
58

sid
58

sid
58

Difference

- *Find skaters that have participated in a local but not in a regional competition*

```
 SELECT P.sid
FROM Participates P, Competition C
WHERE P.cid = C.cid AND C.type = ‘local’
EXCEPT
SELECT P.sid
FROM Participates P, Competition C
WHERE P.cid = C.cid AND C.type = ‘regional’
```

Multiset Semantic in SQL

- Although SQL generally works with multisets, it uses set semantic for union/intersection/difference
- To avoid duplicated elimination for these operators use
  - `UNION ALL, INTERSECT ALL, EXCEPT ALL`
  - Uses multi-set semantic

```
SELECT P.sid
  FROM Participates P, Competition C
  WHERE P.cid = C.cid AND C.type = ‘local’
  UNION ALL
  SELECT P.sid
  FROM Participates P, Competition C
  WHERE  P.cid  =  C.cid  AND  C.type  =  
  ‘regional’
```

Multiset Semantic

- A multiset (bag) may contain the same tuple more than once, although there is no specified order (unlike a list).
  - Example: {1, 2, 1, 3} is a multiset, but not a set
- Multiset Union: {1, 2, 2} $\cup$ {1, 2, 3, 3}
  - Sum the times an element appears in the two multisets
  - Example: {1, 2, 2} $\cup$ {1, 2, 3, 3} = {1, 1, 2, 2, 2, 3, 3}
- Multiset Intersection: {1, 2, 2} $\cap$ {1, 1, 2, 2, 3, 3}
  - Take the minimum of the number of occurrences in each multiset.
  - Example: {1, 2, 2} $\cap$ {1, 1, 2, 2, 3, 3} = {1, 2,2}
- Multiset Difference: {1, 2, 2} - {1, 2, 3, 3}
  - Subtract the number of occurrences in the two multisets
  - Examples: {1, 2, 2} - {1, 2, 3, 3} = {2}
- Some familiar laws for sets also hold for multisets (e.g., union is commutative); but other laws do not hold (e.g., R $\cap$ (S $\cup$ T) $\neq$ (R $\cap$ S) $\cup$ (R $\cap$ T)

Nested queries: The `IN` operator

- A where clause can itself contain an SQL query. The inner query is called a **subquery**
- *Find names of skaters who have particpated in competition #101*

```
SELECT sname
FROM Skaters
WHERE  sid IN (SELECT sid
               FROM Participates
               WHERE cid = 101)
```

- To find skaters who have NOT participated in competition 101 use **`NOT IN`**
- Semantics best understood by nested loop assignment
- Multiple attributes:
  - `WHERE (a1,a2) IN (SELECT a3, a4…`

Non correlated Queries

participates

sid	cid	rank
31	101	2
58	103	7
58	101	7
58	104	1

```
TEMP

SELECT P.sid
FROM Participates P
WHERE P.cid = 101
```

sid
31
58

Not a valid SQL syntax.
Only for demonstrating
The concept. See the previous
Slide for proper SQL.

skaters

sid	sname	rating	age
28	yuppy	9	15
31	debby	7	10
22	conny	5	10
58	lilly	10	13

```
SELECT sname
FROM skaters S
WHERE S.sid IN (SELECT T.sid
                FROM Temp T)
```

sname
debby
lilly

NOT IN

```
SELECT sname
FROM skaters
WHERE sid NOT IN (SELECT sid
                  FROM Participates
                  WHERE cid = 101)
```

sid	sname	rating	age
28	yuppy	9	15
31	debby	7	10
22	conny	5	10
58	lilly	10	13

sid	cid	rank
31	101	2
58	103	7
58	101	7
58	104	1

sid
31
58

sname
yuppy
conny

Multiple attributes

- Seats (<u>stadiumId, seatNum</u>, category)
- Tickets(<u>ticketId</u>, stadiumId, seatNum)
  - stadiumId, seatNum references Seat
- Return all the ticket-ids for Seats of category “gold”

```
SELECT ticketID
FROM Tickets
WHERE (stadiumId, seatNum) IN
    (SELECT stadiumId, seatNum
    FROM Seats
     WHERE category = ‘gold’)
```

Exists Operator

- **`EXISTS`** (relation) is true if the relation is non-empty
- *Find names of skaters who have participated in competition 101*

```
 SELECT S.sname
 FROM Skaters S
 WHERE EXISTS (SELECT *
               FROM Participated P
               WHERE P.cid = 101 AND
                     P.sid = S.sid)
```

- A subquery that refers to values from a surrounding query is called a **correlated subquery.**
- Since the inner query depends on the row of the outer query it must be reevaluated for each row in the outer query

Correlated Query

Find names of skaters who have participated in competition 101

sid	sname	rating	age
28	yuppy	9	15
31	debby	7	10
22	conny	5	10
58	lilly	10	13

sid	cid	rank
31	101	2
58	103	7
58	101	7
58	104	1

```
SELECT sname
FROM skaters s
WHERE EXISTS (SELECT *
              FROM Temp)
```

Not a valid SQL syntax.
Only for demonstrating
The concept. See the previous
Slide for proper SQL.

Correlated Query

Find names of skaters who have participated in competition 101

sid	sname	rating	age
28	yuppy	9	15
31	debby	7	10
22	conny	5	10
58	lilly	10	13

sid	cid	rank
31	101	2
58	103	7
58	101	7
58	104	1

```
SELECT sname
FROM skaters s
WHERE EXISTS (SELECT *
             FROM Temp)
```

Not a valid SQL syntax.
Only for demonstrating
The concept. See the previous
Slide for proper SQL.

sname

Correlated Query

Find names of skaters who have participated in competition 101

sid	sname	rating	age
28	yuppy	9	15
31	debby	7	10
22	conny	5	10
58	lilly	10	13

sid	cid	rank
31	101	2
58	103	7
58	101	7
58	104	1

sname

```
SELECT sname
FROM skaters s
WHERE EXISTS (SELECT *
             FROM Temp)
```

Not a valid SQL syntax.
Only for demonstrating
The concept. See the previous
Slide for proper SQL.

Correlated Query

Find names of skaters who have participated in competition 101

sid	sname	rating	age
28	yuppy	9	15
31	debby	7	10
22	conny	5	10
58	lilly	10	13

sid	cid	rank
31	101	2
58	103	7
58	101	7
58	104	1

sname

Not a valid SQL syntax.
Only for demonstrating
The concept. See the previous
Slide for proper SQL.

```
SELECT sname
FROM skaters s
WHERE EXISTS (SELECT *
              FROM Temp)
```

```
TEMP

SELECT *
FROM Participates P
WHERE P.cid = 101
AND P.sid = 28
```

Correlated Query

Find names of skaters who have participated in competition 101

sid	sname	rating	age
28	yuppy	9	15
31	debby	7	10
22	conny	5	10
58	lilly	10	13

sid	cid	rank
31	101	2
58	103	7
58	101	7
58	104	1

sname

```
SELECT sname
FROM skaters s
WHERE EXISTS (SELECT *
              FROM Temp)
```

Not a valid SQL syntax.
Only for demonstrating
The concept. See the previous
Slide for proper SQL.

Correlated Query

Find names of skaters who have participated in competition 101

sid	sname	rating	age
28	yuppy	9	15
31	debby	7	10
22	conny	5	10
58	lilly	10	13

sid	cid	rank
31	101	2
58	103	7
58	101	7
58	104	1

Not a valid SQL syntax.
Only for demonstrating
The concept. See the previous
Slide for proper SQL.

sname

```
SELECT sname
FROM skaters s
WHERE EXISTS (SELECT *
              FROM Temp)
```

```
TEMP

SELECT *
FROM Participates P
WHERE P.cid = 101
AND P.sid = 31
```

Correlated Query

Find names of skaters who have participated in competition 101

sid	sname	rating	age
28	yuppy	9	15
31	debby	7	10
22	conny	5	10
58	lilly	10	13

sid	cid	rank
31	101	2
58	103	7
58	101	7
58	104	1

Not a valid SQL syntax.
Only for demonstrating
The concept. See the previous
slide for proper SQL.

```
SELECT sname
FROM skaters s
WHERE EXISTS (SELECT *
              FROM Temp)
```

TEMP

```
SELECT *
FROM Participates P
WHERE P.cid = 101
AND P.sid = 31
```

sname
debby

Correlated Query

Find names of skaters who have participated in competition 101

sid	sname	rating	age
28	yuppy	9	15
31	debby	7	10
22	conny	5	10
58	lilly	10	13

sid	cid	rank
31	101	2
58	103	7
58	101	7
58	104	1

Not a valid SQL syntax. Only for demonstrating The concept. See the previous slide for proper SQL.

sname
debby

```
SELECT sname
FROM skaters s
WHERE EXISTS (SELECT *
             FROM Temp)
```

Correlated Query

Find names of skaters who have participated in competition 101

sid	sname	rating	age
28	yuppy	9	15
31	debby	7	10
22	conny	5	10
58	lilly	10	13

sid	cid	rank
31	101	2
58	103	7
58	101	7
58	104	1

Not a valid SQL syntax.
Only for demonstrating
The concept. See the
previous slide for proper SQL.

sname
debby

```
SELECT sname
FROM skaters s
WHERE EXISTS (SELECT *
              FROM Temp)
```

TEMP

```
SELECT *
FROM Participates P
WHERE P.cid = 101
s.sid= 22
```

Correlated Query

Find names of skaters who have participated in competition 101

sid	sname	rating	age
28	yuppy	9	15
31	debby	7	10
22	conny	5	10
58	lilly	10	13

sid	cid	rank
31	101	2
58	103	7
58	101	7
58	104	1

```
SELECT sname
FROM skaters s
WHERE EXISTS (SELECT *
             FROM Temp)
```

Not a valid SQL syntax. Only for demonstrating The concept. See the previous slide for proper SQL.

sname
debby

Correlated Query

Find names of skaters who have participated in competition 101

sid	sname	rating	age
28	yuppy	9	15
31	debby	7	10
22	conny	5	10
58	lilly	10	13

sid	cid	rank
31	101	2
58	103	7
58	101	7
58	104	1

Not a valid SQL syntax. Only for demonstrating The concept. See the previous slide for proper SQL.

sname
debby

```
SELECT sname
FROM skaters s
WHERE EXISTS (SELECT *
              FROM Temp)
```

TEMP

```
SELECT *
FROM Participates P
WHERE P.cid = 101
s.sid= 58
```

Correlated Query

Find names of skaters who have participated in competition 101

sid	sname	rating	age
28	yuppy	9	15
31	debby	7	10
22	conny	5	10
58	lilly	10	13

sid	cid	rank
31	101	2
58	103	7
58	101	7
58	104	1

Not a valid SQL syntax. Only for demonstrating The concept. See the previous slide for proper SQL.

sname
debby
lilly

```
SELECT sname
FROM skaters s
WHERE EXISTS (SELECT *
             FROM Temp)
```

TEMP

```
SELECT *
FROM Participates P
WHERE P.cid = 101
s.sid= 58
```

Quantifiers

- **`ANY`** and **`ALL`** behave as existential and universal quantifiers, respectively.
- Syntax
  - `WHERE attr op ANY (SELECT …`
  - `WHERE attr op ALL (SELECT`
  - `op` is one of `<, =, >, <>, <=, >=`
- *Find the skater with the highest rating*

```
SELECT *
FROM Skaters
WHERE rating >= ALL (SELECT rating
                     FROM Skaters)
```

sid	sname	rating	age
28	yuppy	9	15
31	debby	7	10
22	conny	5	10
58	lilly	10	13

Rating
9
7
5
10

Complex queries

What do the following two queries return?

```
  SELECT sname
  FROM Skaters S
  WHERE NOT EXISTS ((SELECT C.cid
                       FROM Competition C)
                       EXCEPT
                     (SELECT P.cid
                      FROM Participates P
                      WHERE P.sid=S.sid))
```

- The inner correlated query (executed for each skater in the outer table) returns all competitions in which the skater did not compete
- The WHERE NOT EXISTS is true whenever the result of the inner query is empty; that is, if the skater has competed in all competitions.
- Thus, the query returns skaters that have competed in all competitions

```
SELECT sname
FROM Skaters S
WHERE NOT EXISTS (SELECT C.cid
                  FROM Competition C
                  WHERE NOT EXISTS
                       (SELECT P.cid
                        FROM Participates P
                        WHERE P.cid = C.cid
                        AND P.sid = S.sid))
```

Equivalent to the above inner Except query.

- The most inner subquery returns for each competition c of the middle query the cid of c if the skater of the most outer table has competed in c
- The "middle" query returns the competitions in which the skater has not competed
- The most outer "NOT EXIST" is true when there are no competitions in which the skater has not competed
- Thus, the query returns skaters that have competed in all competitions.