Ch 5. SQL: Queries, Constraints, Triggers

Sang-Won Lee

http://icc.skku.ac.kr/~swlee

SKKU VLDB Lab.

(http://vldb.skku.ac.kr/)

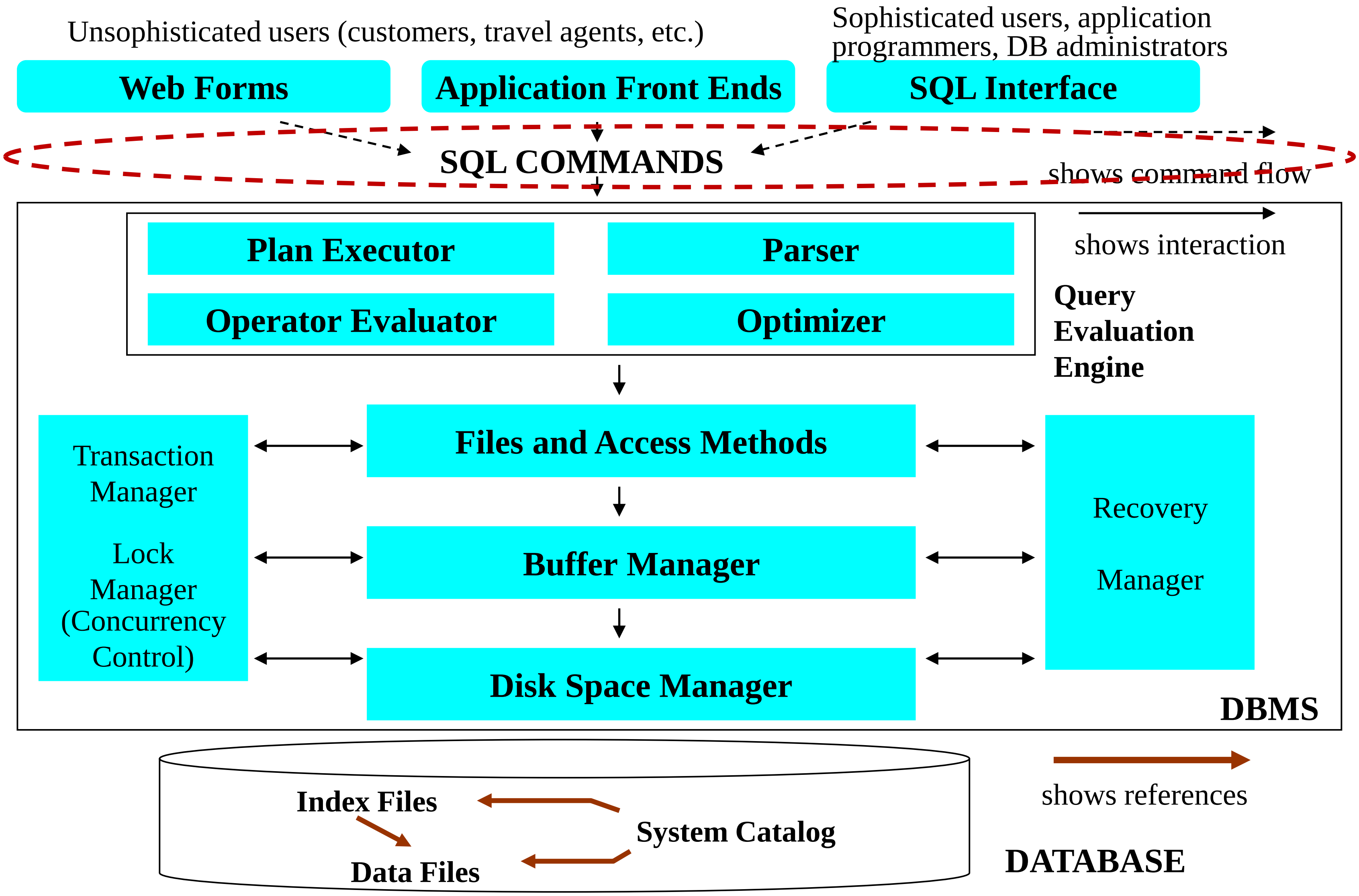
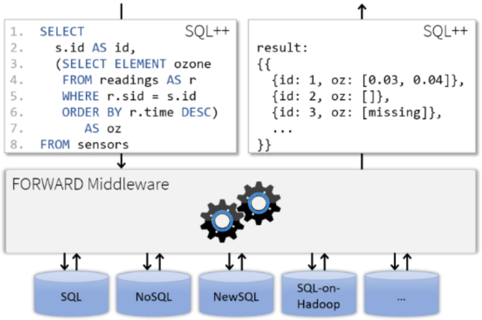


Figure 1.3 Anatomy of an RDBMS

SQL in Big Data Era

- What is SQL?
  - DB dead? (5 Years ago)
- The BIG data era has come
- Now everyone sells SQL
  - SQL-on-{Hadoop, NoSQL}
  - Google's cloud Spanner, BiqQuery
  - Amazon's RDS, AuroraDB
  - Databricks's DeltaLake, Snowflake
- SQL becomes the lingua franca of the data management world in big data era?!



Very Large Data Bases

Overview

<u>SQL Expressive Powers</u>

1. Relational Algebra or Calculus
2. Aggregation / Grouping
3. Deductive Logics / Analytic Functions (Windowing)
4. Data Mining Features
5. **Machine Learning (e.g. + Linear Albegra?)**

- DML: queries + insert/update/delete
- DDL(Data Definition Language)
  - Logical: table/view/ICs
  - Physical: index/partitions
- DCL(Data Control Language)
  - Trigger, Advanced ICs, Transaction Management, Security
- Misc.
  - Embedded/Dynamic SQL, Client-Server/Remote DB Access (e.g. ODBC/JDBC)
  - Advanced Features: OO, <u>Logics(recursive queries), DSS & OLAP, Data Mining</u>, Spatial/Temporal DB, Text & XML Data, Game, Social Network

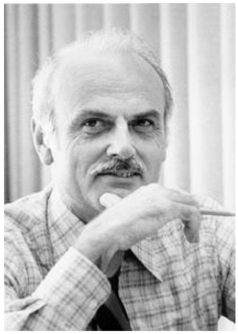
** Refer to 5.1 for details

SQL and Data Independence

- "Queries should be expressed in terms of high-level, nonprocedural concepts that are independent of physical representation. Selection of an algorithm for processing a given query could then be done by an optimizing compiler, based on the access paths available and the statistics of the stored data; if these access paths or statistics should later change, the algorithm could be re-optimized without human intervention." (SQL @ Encyclopedia of Database Systems)

- Design considerations of relational data model (and RDBMS) for data independence
  - 1) High-level language (declarative, non-procedural SQL), 2) value-based relationship, 3) content(or name)-based addressing (e.g. unlike (x,y) in Excel or C-like pointer, detpno =10, column C instead of in 3rd column), and 4) no assumptions about physical schema such as data layout (row-wise vs. column-wise) , existence of indexes and table partitioning

- SQL's Benefits
  - Database application **development productivity**
  - Data Independence



5.2 Example Schemas

Boats

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

Sailors

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

Reserves

sid	bid	day
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

M:N relationship
b/w Sailors and Boats

- Tables and queries from http://www.cs.wisc.edu/~dbbook

SQL Expressive Powers

1. Relational Algebra or Calculus
2. Aggregation / Grouping
3. Deductive Logics / Analytic Functions (Windowing)
4. Data Mining Features

Basic SQL Query

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

- ***relation-list*** A list of relation names, possibly with a *range (or tuple)* variable after each name.
- ***target-list*** A list of attributes of relations in *relation-list*
- ***qualification*** Comparisons (attr *op* const or attr1 *op* attr2, where *op* is one of $<, >, =, \leq, \geq, \neq$) combined using logical connectives AND, OR, and NOT.
- DISTINCT is an optional keyword indicating that the answer should not contain duplicates.
  - Note: Set semantics in R.A. vs. multi-set(or bag) semantics in SQL
  - By default, duplicates are **NOT** eliminated in SQL.

Single Table Query

SELECT S.sid, S.sname, S.rating, S.age
FROM Sailors AS S / S: tuple variable */*
WHERE S.rating > 7

SELECT S.name, S.age
FROM Sailors (AS) S

SELECT DISTINCT S.name, S.age
FROM Sailors (AS) S

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

sname	age
Dustin	45.0
Brutus	33.0
Lubber	55.5
Andy	25.5
Rusty	35.0
Horatio	35.0
Zorba	16.0
Art	25.5
Bob	63.5

sname	age
Dustin	45.0
Brutus	33.0
Lubber	55.5
Andy	25.5
Rusty	35.0
Horatio	35.0
Zorba	16.0
Horatio	35.0
Art	25.5
Bob	63.5

Conceptual Evaluation Strategy

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  1. Compute cross-product of *relation-list*. (what if any relation is empty?)
  2. Discard resulting tuples if they fail *qualifications* *(i.e., NOT TRUE)*.
  3. Delete attributes that are not in *target-list*.
  4. 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*.

Example of Conceptual Evaluation

Q1: Find the names of sailors who reserved boat 103

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

sid	*bid*	*day*
22	101	10/10/96
58	103	11/12/96

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

1. FROM

2. WHERE

3. SELECT

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

A Note on Range Variables (or Tuple Variables)

- Really needed only if the same relation appears twice in the FROM clause (that is, self-join). The previous query can also be written as:

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

OR

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

> **Tuple Variables**
>
> Sometimes we need to refer to two or more tuples in the same relation. To do so, we define several tuple variables for that relation in the from clause and use the tuple variables as aliases of the relation. The effect is exactly the same as was achieved by the range-statement in QUEL, so SQL, which appeared at first to be a "syntactically sugared" form of relational algebra, is now revealed to resemble tuple relational calculus.

- It is good style, however, to use range (or tuple) variable always!

Find the name of sailor who reserved boat 103

- SQL (by user)

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

- Tuple Relational Calculus (TRC)

$$\{P \mid \exists S \in Sailors \;\; \exists R \in Reserves(R.sid = S.sid \wedge R.bid = 103$$
$$\wedge P.sname = S.sname)\}$$

- Relational Algebra: 2 different expressions

1. DBMS translate SQL into relational algebra

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

2. Query optimizer (QO) looks for other algebra expressions that produce the same result
3. QO will find the second expression is likely to be less expensive because the sizes of intermediate relations are smaller, thanks to the early use of selection

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

Very Large Data Bases

Q4: Find sailors who've reserved at least one boat

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

sid	*bid*	*day*
22	101	10/10/96
58	103	11/12/96

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

- Join can be interpreted as existential quantifier
- Would adding DISTINCT to this query make a difference?
- What is the effect of replacing *S.sid* by *S.sname* in the SELECT clause? Would adding DISTINCT to this variant of the query make a difference?

5.2.2 Expressions and Strings

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

- Illustrates use of arithmetic expressions and string pattern matching: *Find triples (of ages of sailors and two fields defined by expressions) for sailors whose names begin and end with B and contain at least three characters.*
- AS and = are two ways to name fields in result.
- LIKE is used for string matching. `_' stands for any one character and `%' stands for 0 or more arbitrary characters.
- Date and time format: e.g. Oracle (http://infolab.stanford.edu/~ullman/fcdb/oracle/or-time.html)

Regular Expressions in SQL

```
SELECT zip
FROM zipcode
WHERE REGEXP_LIKE(zip, '[^[:digit:]]')

ZIP
-----
ab123
123xy
007ab
abcxy
```

- The increased importance of text data → Regular Expression
  - See here for Regular Expression in Oracle 10g
    - ✓ Here: http://docs.oracle.com/cd/B19306_01/appdev.102/b14251/adfns_regexp.htm

5.3 Union, Intersect, Except

- Another basic operators in R.A
  - UNION, DIFFERENCE

- ANSI SQL provides the following set operators
  - UNION, UNION ALL
  - INTERSECT, INTERSECT ALL
  - EXCEPT(or MINUS)
  - IN, ANY, ALL, EXISTS (covered in Section 5.4)

Q5: Find sid's of sailors who've reserved a red <u>or</u> a green boat

- UNION: Can be used to compute the union of any two *union-compatible* sets of tuples (which are themselves the result of SQL queries).

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

- If we replace OR by AND in the first version, what do we get?

- Also available: EXCEPT (What do we get if we replace UNION by EXCEPT?)

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

Q6: Find sid's of sailors who've reserved a red <u>and</u> a green boat

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

- INTERSECT: Can be used to compute the intersection of any two *union-compatible* sets of tuples.
- Included in the SQL/92 standard, but some systems don't support it.
- Contrast symmetry of the UNION and INTERSECT queries with how much the other versions differ.

Key field!

```
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'
```

5.4 Nested Queries(or Subqueries)

Q1: Find names of sailors who've reserved boat #103:

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

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

- A very powerful feature of SQL
  - a WHERE clause can itself contain an SQL query!
  - Also, SELECT, FROM, or HAVING clause can.
- 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.*
- Do the two queries above have the same semantic?
  - What if one sailor reserved two or more boats?

Multiply Nested Queries

Q2: Find names of sailors who have (not) reserved a red boat:

```
SELECT  S.sname
FROM  Sailors S
WHERE  S.sid (NOT) IN  (SELECT  R.sid
                        FROM  Reserves R
                        WHERE  R.bid IN ( SELECT B.bid
                                          FROM Boats B
                                          WHERE B.color = 'red')
```

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

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

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

Nested Queries with Correlation

Q1: 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, like IN.
- Correlated vs. non-correlated: illustrates why, in general, subquery must be re-computed for each Sailors tuple.
- If UNIQUE is used, and * is replaced by *R.bid*, finds sailors with at most one reservation for boat #103. (UNIQUE checks for duplicate tuples; * denotes all attributes. Why do we have to replace * by *R.bid*? Because "unique(select * .." will return True even though more than one reservation for boat 103 exists!)

More on Set-Comparison Operators

- We've seen IN, EXISTS and UNIQUE. Also, NOT IN, NOT EXISTS and NOT UNIQUE.
- Also: *op* ANY(SOME), *op* ALL
  - op: $>, <, =, \geq, \leq, \neq$
  - IN equivalent to =ANY; NOT IN equivalent to <> ALL
- e.g. Find sailors whose rating is greater than that of some sailors called Horatio:

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

- What if inner query is empty? False
- ANY → ALL: True

Very Large Data Bases

Rewriting INTERSECT Queries Using IN

*Q6: Find sid's of sailors who've reserved both a red **and** a green boat:*

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

Who reserved Red Boat

And, further who has sids that is in Green Boat Reservers list

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

Division in SQL

Q9: Find sailors who've reserved all boats.

(1)

```
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))
```

- Let's do it the hard way, without EXCEPT:

(2)

```
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))
```

Sailors S such that ..

there is no boat B without ..

a Reserves tuple showing S reserved B

Very Large Data Bases

SQL Expressive Powers
1. Relational Algebra or Calculus
2. Aggregation / Grouping
3. Deductive Logics / Analytic Functions (Windowing)
4. Data Mining Features

5.5 Aggregate Operators

- Significant extension of Relational Algebra

```
COUNT (*)
COUNT ( [DISTINCT] A)
SUM ( [DISTINCT] A)
AVG ( [DISTINCT] A)
MAX (A)
MIN (A)
```

single column

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

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

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

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

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

Find name and age of the oldest sailor(s)

- The first query is illegal! (We'll look into the reason a bit later, when we discuss GROUP BY.)

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

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

- The third query is equivalent to the second query, and is allowed in the SQL/92 standard, but is not supported in some systems .

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

Aggregate Operators: Any / All Alternatives

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

```
SELECT  S.sname
FROM  Sailors S
WHERE S.age > ALL (SELECT S2.age
                   FROM Sailors S2
                   WHERE S2.rating = 10)
```

GROUP BY and HAVING

- So far, we've applied aggregate operators to all (qualifying) tuples. In many real-world scenarios, however, we want to apply them to each of several ***groups*** of tuples.
- *Q31: 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 .. 10
    SELECT  MIN (S.age)
    FROM  Sailors S
    WHERE  S.rating = i
```

  - In general, we don't know how many rating levels exist, and what the rating values for these levels are!

Group By: Examples

- For each rating, find the average age of the sailors

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

- For each rating, find the age of the youngest sailor with age $\geq$ 18

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

Queries With 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 (i.e. attribute list) (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

1. The cross-product of *relation-list* is computed, tuples that fail *qualification* are discarded,

2. `*unnecessary'* fields are deleted, and the remaining tuples are partitioned into groups by the value of attributes in *grouping-list*.

3. The *group-qualification* is then applied to eliminate some groups.(by HAVING clause)

4. One answer tuple per qualifying group is generated.

Very Large Data Bases

Example(1)

SELECT S.rating, MIN (S.age)
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating

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

1. Form cross product

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

2. Delete unneeded columns, rows; form groups

3. Perform Aggregation

rating	age
1	33.0
7	35.0
8	55.0
10	35.0

Answer Table

Example(2) Q32: Find the age of the youngest sailor with age >= 18, for each rating with at least 2 <u>such</u> 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
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
96	Frodo	3	25.5

rating	*age*
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
9	35.0
3	25.5
3	63.5
3	25.5

1) After WHERE

rating	*age*
1	33.0
3	25.5
3	25.5
3	63.5
7	45.0
7	35.0
8	55.5
8	25.5
9	35.0
10	35.0

2) After GROUP BY

Answer relation

rating	*minage*
3	25.5
7	35.0
8	25.5

3) After HAVING and AGG.

Q33: For each red boat, find # of reservations for this boat

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

What if a red boat has no reservation?

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

ILLEGAL!! WHY??

Q34: Find the average age for each rating that has at least 2 sailors (of any age)

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

rating	*avgage*
3	44.5
7	40.0
8	40.5
10	25.5

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
96	Frodo	3	25.5

Q35: Find the average age of sailors with age > 18, for each rating with at least 2 sailors (of any age)

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

- Shows HAVING clause can also contain a subquery.
- Compare this with the query (Q32) where we considered only ratings with 2 sailors over 18!
  - HAVING COUNT(*) >1

Very Large Data Bases

Q37: Find those ratings for which the average age is the minimum over all ratings

- Aggregate operations cannot be nested!

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

Cf. Oracle allows nested aggregation.

- Correct solution (in SQL/92 and Oracle):

```
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)
```

```
WITH Temp AS (  -- Subquery factoring in Oracle
        SELECT S.rating, avg(S.sal) as avgsal
        FROM  Sailors S
        GROUP BY S.rating )
SELECT Temp.rating, Temp.avgage
FROM Temp
WHERE Temp.avgsal = (SELECT  MIN(Temp.agvage)
                              FROM Temp)
```

5.6 Null Values

- Field values in a tuple are sometimes *unknown* (e.g., a rating has not been assigned) or *inapplicable* (e.g., no spouse's name).
  - SQL provides a special value *null* for such situations.

- The presence of *null* complicates many issues.

- Optional reading: Nulls: nothing to worry about

Null Values(2)

1. Special operators needed to check if value is/is not *null*.
   - IS NULL, IS NOT NULL
   - NVL(attr, const) function

2. Is *rating>8* true or false when *rating IS NULL*? What is the result of AND, OR and NOT logical connectives involving NULL? We need a 3-valued logic (true, false and *unknown; T, F, U*)
   - If rating is null, rating > 8 is evaluated to “U”, not “T” or “F”
   - Null = Null ➔ U
   - NOT (U) = U
   - T OR U = T; F OR U = U; F AND xx = F; U AND (T or U) = U

Null Values(3)

3. Meaning of constructs must be defined carefully.
   - WHERE clause eliminates rows that don't evaluate to **true**. That is, false and unknown tuple is discarded!!
   - Duplicate semantics: duplicate if corresponding columns has nulls
   - +, -, *,/ returns null if one of arguments is null
   - COUNT(*) counts null values; all other aggregate operations simply ignore nulls!!
     - ✓ AVG/SUM/MIN/MAX of nulls → null
     - ✓ Count(attr) also ignores null value

Null Values(4)

4. New operators (in particular, outer joins) possible/needed.

sid	*bid*	*day*
22	101	10/10/96
58	103	11/12/96

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

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

sid	*bid*
22	101
~~31~~	~~*null*~~
58	103

INNER JOIN

SELECT S.sid, R.bid
FROM Sailors S, Reserves R
WHERE S.sid = R.sid(+)

sid	*bid*
22	101
31	*null*
58	103

OUTER JOIN

Null Values(5)

- ANSI join syntax

```
SELECT  S.sid, R.bid
FROM  Sailors S, Reserves R
WHERE  S.sid = R.sid and S.age > 18 and R.date = '04/03/24'
```

JOIN condition　　SELECTION condition

```
SELECT  S.sid, R.bid
FROM  Sailors S NATURAL JOIN Reserves R
WHERE  S.age > 18 and R.date = '04/03/24'
```

- See Oracle SQL Language Reference for ANSI JOIN in Oracle

Very Large Data Bases

Null Values(6)

5. Disallowing NULL values
   - sname CHAR(20) NOT NULL
   - implicit NOT NULL in Primary key constraint!
   - Column with UNIQUE constraint is NULLable; two or more tuples can have null values for the column
6. IN & NOT IN
   - 1 in {1, null}: True,   2 in {1, null} : False
     - ✓ Thus, in **==** exists
   - 1 not in {1, null}: False,   2 not in {1, null} : False
     - ✓ Thus, not in **!=** not exists

5.7 Integrity Constraints (Review)

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

General Constraints

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

Per-tuple Check

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

Intra- or Inter-Table Check

```
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)))
```

5.7.3 Assertions: ICs over Multiple Relations

- Awkward and wrong!
- If Sailors is empty, the number of Boats tuples can be anything! – why?
- Assertion is the right solution; not associated with either table.

```
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 )
```

(# of boats + # of Sailors) < 100

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

5.8 Triggers and Active Database

- Trigger: procedure that starts automatically if specified changes occur to the DBMS → ACTIVE Database!!
- Three parts: ECA Rule
  - Event (activates the trigger)
  - Condition (tests whether the triggers should run)
  - Action (what happens if the trigger runs)

```
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
```

Triggers

- Read Sec. 5.9.3
- See here for trigger example in Oracle 10g
  - Provide sophisticated auditing
  - Prevent invalid transactions
  - Enforce referential integrity (either those actions not supported by declarative integrity constraints or across nodes in a distributed database)
  - Enforce complex business rules
  - Enforce complex security authorizations
  - Provide transparent event logging
  - .....

Summary

- SQL was an important factor in the early acceptance of the relational model; more natural than earlier, procedural query languages, more productive!!.
- Relationally complete; in fact, significantly more expressive power than relational algebra.
- Even queries that can be expressed in RA can often be expressed more naturally in SQL.
- Many alternative ways to write a query; optimizer should look for most efficient evaluation plan.
  - In practice, users need to be aware of how queries are optimized and evaluated for best results.
- NULL for unknown field values brings many complications
- SQL allows specification of rich integrity constraints
- Triggers respond to changes in the database