Review
SQL

Purposes of SQL

- Data Manipulation Language (DML)
  - Querying: SELECT-FROM-WHERE
  - Modifying: INSERT/DELETE/UPDATE
- Data Definition Language (DDL)
  - CREATE/ALTER/DROP

The SQL Query Language

- To find all 18 year old students, we can write:

all attributes

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

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2

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

```
SELECT S.name, S.login
```

Querying Multiple Relations

- What does the following query compute?

```
SELECT S.name, E.cid
FROM Students S, Enrolled E
WHERE S.sid=E.sid AND E.grade="A"
```

Given the following instances of Enrolled and Students:

Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Enrolled

sid	cid	grade
53831	Carnatic101	C
53831	Reggae203	B
53650	Topology112	A
53666	History105	B

we get: ??

Querying Multiple Relations

- What does the following query compute?

```
SELECT S.name, E.cid
FROM Students S, Enrolled E
WHERE S.sid=E.sid AND E.grade="A"
```

Given the following instances of Enrolled and Students:

Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Enrolled

sid	cid	grade
53831	Carnatic101	C
53831	Reggae203	B
53650	Topology112	A
53666	History105	B

we get:

S.name	E.cid
Smith	Topology112

Creating Relations in SQL

- Creates the "Students" relation
  - the type (domain) of each field is specified
  - enforced by the DBMS whenever tuples are added or modified

```
CREATE TABLE Students
    (sid CHAR(20),
     name CHAR(20),
     login  CHAR(10),
     age INTEGER,
     gpa REAL)
```

- As another example, the "Enrolled" table holds information about courses that students take

```
CREATE TABLE Enrolled
    (sid CHAR(20),
     cid CHAR(20),
     grade CHAR(2))
```

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Students

sid	cid	grade
53831	Carnatic101	C
53831	Reggae203	B
53650	Topology112	A
53666	History105	B

Enrolled

Destroying and Altering Relations

```
DROP TABLE  Students
```

- Destroys the relation Students
  - The schema information *and* the tuples are deleted.

```
ALTER TABLE  Students
    ADD COLUMN firstYear: integer
```

- The schema of Students is altered by adding a new field; every tuple in the current instance is extended with a NULL value in the new field.

Adding and Deleting Tuples

- Can insert a single tuple using:

```
INSERT INTO  Students (sid, name, login, age, gpa)
VALUES (53688, 'Smith', 'smith@ee', 18, 3.2)
```

- Can delete all tuples satisfying some condition (e.g., name = Smith):

```
DELETE
FROM Students S
WHERE S.name = 'Smith'
```

Integrity Constraints (ICs)

- IC: condition that must be true for any instance of the database
  - e.g., domain constraints
  - ICs are specified when schema is defined
  - ICs are checked when relations are modified
- A legal instance of a relation is one that satisfies all specified ICs
  - DBMS will not allow illegal instances
- If the DBMS checks ICs, stored data is more faithful to real-world meaning
  - Avoids data entry errors, too!

Keys in a Database

- Key / Candidate Key
- Primary Key
- Foreign Key

- Primary key attributes are underlined in a schema
  - Person(pid, address, name)
  - Person2(address, name, age, job)

Primary Key Constraints

- A set of fields is a key for a relation if :
  1. No two distinct tuples can have same values in all key fields, and
  2. This is not true for any subset of the key

- If there are > 1 keys for a relation, one of the keys is chosen (by DBA = DB admin) to be the primary key
  - E.g., sid is a key for Students
  - The set {sid, gpa} is a superkey.

- Any possible benefit to refer to a tuple using primary key (than any key)?

Primary and Candidate Keys in SQL

- Possibly many candidate keys
  - specified using UNIQUE
  - one of which is chosen as the primary key.

- "For a given student and course, there is a single grade."

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  ???)
```

Primary and Candidate Keys in SQL

- Possibly many candidate keys
  - specified using UNIQUE
  - one of which is chosen as the primary key.

- "For a given student and course, there is a single grade."

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  (sid, cid) )
```

Primary and Candidate Keys in SQL

- Possibly many candidate keys
  - specified using UNIQUE
  - one of which is chosen as the primary key.

- "For a given student and course, there is a single grade."

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  (sid, cid) )
```

- vs.

- "Students can take only one course, and receive a single grade for that course; further, no two students in a course receive the same grade."

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY ???,
   UNIQUE ??? )
```

Primary and Candidate Keys in SQL

- Possibly many candidate keys
  - specified using UNIQUE
  - one of which is chosen as the primary key.

- "For a given student and course, there is a single grade."
- vs.
- "Students can take only one course, and receive a single grade for that course; further, no two students in a course receive the same grade."

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  (sid,cid) )
```

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY sid,
   UNIQUE (cid, grade))
```

Primary and Candidate Keys in SQL

- Possibly many candidate keys
  - specified using UNIQUE
  - one of which is chosen as the primary key.

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  (sid,cid) )
```

- "For a given student and course, there is a single grade."
- vs.
- "Students can take only one course, and receive a single grade for that course; further, no two students in a course receive the same grade."
- Used carelessly, an IC can prevent the storage of database instances that arise in practice!

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY sid,
   UNIQUE (cid, grade))
```

Foreign Keys, Referential Integrity

- Foreign key : Set of fields in one relation that is used to `refer' to a tuple in another relation
  - Must correspond to primary key of the second relation
  - Like a `logical pointer'

- E.g. sid is a foreign key referring to Students:
  - Enrolled(sid: string, cid: string, grade: string)
  - If all foreign key constraints are enforced, referential integrity is achieved
  - i.e., no dangling references

Foreign Keys in SQL

- Only students listed in the Students relation should be allowed to enroll for courses

```
CREATE TABLE Enrolled
  (sid CHAR(20), cid CHAR(20), grade CHAR(2),
   PRIMARY KEY  (sid,cid),
   FOREIGN KEY (sid) REFERENCES Students )
```

Enrolled

sid	cid	grade
53666	Carnatic101	C
53666	Reggae203	B
53650	Topology112	A
53666	History105	B

Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Enforcing Referential Integrity

- Consider Students and Enrolled
  - sid in Enrolled is a foreign key that references Students.
- What should be done if an Enrolled tuple with a non-existent student id is inserted?
  - Reject it!
- What should be done if a Students tuple is deleted?
  - Three semantics allowed by SQL
  1. Also delete all Enrolled tuples that refer to it (cascade delete)
  2. Disallow deletion of a Students tuple that is referred to
  3. Set sid in Enrolled tuples that refer to it to a default sid
  4. (in addition in SQL): Set sid in Enrolled tuples that refer to it to a special value null, denoting `unknown' or `inapplicable'
- Similar if primary key of Students tuple is updated

Referential Integrity in SQL

- SQL/92 and SQL:1999 support all 4 options on deletes and updates.
  - Default is NO ACTION (delete/update is rejected)
  - CASCADE (also delete all tuples that refer to deleted tuple)
  - SET NULL / SET DEFAULT (sets foreign key value of referencing tuple)

```
CREATE TABLE Enrolled
  (sid CHAR(20) DEFAULT '000',
   cid CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  (sid,cid),
   FOREIGN KEY (sid)
    REFERENCES Students
     ON DELETE CASCADE
     ON UPDATE SET DEFAULT )
```

Where do ICs Come From?

- ICs are based upon the semantics of the real-world enterprise that is being described in the database relations

- Can we infer ICs from an instance?
  - We can check a database instance to see if an IC is violated, but we can NEVER infer that an IC is true by looking at an instance.
  - An IC is a statement about all possible instances!
  - From example, we know name is not a key, but the assertion that sid is a key is given to us.

- Key and foreign key ICs are the most common; more general ICs supported too

Example Instances

- What does the key (sid, bid, day) in Reserves mean?
- If the key for the Reserves relation contained only the attributes *(sid, bid)*, how would the semantics differ?

Sailor

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

Reserves

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

Basic SQL Query

```
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
- qualification Comparisons
  - (Attr op const) or (Attr1 op Attr2)
  - where op is one of = , <, >, <=, >= combined using AND, OR and NOT
- DISTINCT is an optional keyword indicating that the answer should not contain duplicates
  - Default is that duplicates are not eliminated!

Can see the next few slides first

Conceptual Evaluation Strategy

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

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  - Compute the cross-product of *<relation-list>*
  - 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

Example of Conceptual Evaluation

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

Sailor

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

Step 1: Form **cross product** of Sailor and Reserves

sid	sname	rating	age	sid	bid	day
22	dustin	7	45	22	101	10/10/96
22	dustin	7	45	58	103	11/12/96
31	lubber	8	55	22	101	10/10/96
31	lubber	8	55	58	103	11/12/96
58	rusty	10	35	22	101	10/10/96
58	rusty	10	35	58	103	11/12/96

Reserves

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

Example of Conceptual Evaluation

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

Sailor

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

Step 2: Discard tuples that do not satisfy <qualification>

sid	sname	rating	age	sid	bid	day
~~22~~	~~dustin~~	~~7~~	~~45~~	~~22~~	~~101~~	~~10/10/96~~
~~22~~	~~dustin~~	~~7~~	~~45~~	~~58~~	~~103~~	~~11/12/96~~
~~31~~	~~lubber~~	~~8~~	~~55~~	~~22~~	~~101~~	~~10/10/96~~
~~31~~	~~lubber~~	~~8~~	~~55~~	~~58~~	~~103~~	~~11/12/96~~
~~58~~	~~rusty~~	~~10~~	~~35~~	~~22~~	~~101~~	~~10/10/96~~
58	rusty	10	35	58	103	11/12/96

Reserves

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

Example of Conceptual Evaluation

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

Sailor

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

Reserves

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

Step 3: Select the specified attribute(s)

sid	sname	rating	age	sid	bid	day
~~22~~	~~dustin~~	~~7~~	~~45~~	~~22~~	~~101~~	~~10/10/96~~
~~22~~	~~dustin~~	~~7~~	~~45~~	~~58~~	~~103~~	~~11/12/96~~
~~31~~	~~lubber~~	~~8~~	~~55~~	~~22~~	~~101~~	~~10/10/96~~
~~31~~	~~lubber~~	~~8~~	~~55~~	~~58~~	~~103~~	~~11/12/96~~
~~58~~	~~rusty~~	~~10~~	~~35~~	~~22~~	~~101~~	~~10/10/96~~
58	rusty	10	35	58	103	11/12/96

A Note on "Range Variables"

- Really needed only if the same relation appears twice in the FROM clause
  - sometimes used as a short-name
- 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
```

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

Joins

- Condition/Theta-Join
- Equi-Join
- Natural-Join
- (Left/Right/Full) Outer-Join

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

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

Condition/Theta Join

```
SELECT *
FROM Sailors S, Reserves R
WHERE S.sid=R.sid and age >= 40
```

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

Form cross product, discard rows that do not satisfy the condition

sid	sname	rating	age	sid	bid	day
22	dustin	7	45	22	101	10/10/96
~~22~~	~~dustin~~	~~7~~	~~45~~	~~58~~	~~103~~	~~11/12/96~~
31	lubber	8	55	22	101	10/10/96
~~31~~	~~lubber~~	~~8~~	~~55~~	~~58~~	~~103~~	~~11/12/96~~
~~58~~	~~rusty~~	~~10~~	~~35~~	~~22~~	~~101~~	~~10/10/96~~
58	rusty	10	35	58	103	11/12/96

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

Equi Join

```
SELECT *
FROM Sailors S, Reserves R
WHERE S.sid=R.sid and age = 45
```

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

A special case of theta join
Join condition only has equality predicate =

sid	sname	rating	age	sid	bid	day
22	dustin	7	45	22	101	10/10/96
~~22~~	~~dustin~~	~~7~~	~~45~~	~~58~~	~~103~~	~~11/12/96~~
~~31~~	~~lubber~~	~~8~~	~~55~~	~~22~~	~~101~~	~~10/10/96~~
~~31~~	~~lubber~~	~~8~~	~~55~~	~~58~~	~~103~~	~~11/12/96~~
~~58~~	~~rusty~~	~~10~~	~~35~~	~~22~~	~~101~~	~~10/10/96~~
~~58~~	~~rusty~~	~~10~~	~~35~~	~~58~~	~~103~~	~~11/12/96~~

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

Natural Join

```
SELECT *
FROM Sailors S NATURAL JOIN Reserves R
```

A special case of equi join
Equality condition on ALL common predicates (sid)
Duplicate columns are eliminated

sid	sname	rating	age	bid	day
22	dustin	7	45	101	10/10/96
~~22~~	~~dustin~~	~~7~~	~~45~~	~~103~~	~~11/12/96~~
~~31~~	~~lubber~~	~~8~~	~~55~~	~~101~~	~~10/10/96~~
~~31~~	~~lubber~~	~~8~~	~~55~~	~~103~~	~~11/12/96~~
~~58~~	~~rusty~~	~~10~~	~~35~~	~~101~~	~~10/10/96~~
58	rusty	10	35	103	11/12/96

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

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

Outer Join

```
SELECT S.sid, R. bid
FROM Sailors S LEFT OUTER JOIN Reserves R
ON  S.sid=R.sid
```

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

Preserves all tuples from the left table whether or not there is a match
if no match, fill attributes from right with null
Similarly RIGHT/FULL outer join

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

sid	bid
22	101
31	null
58	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*
- LIKE is used for string matching. `_' stands for any one character and `%' stands for 0 or more arbitrary characters
  - You will need these often

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

Sailors (sid, sname, rating, age)
Reserves(sid, bid, day)
Boats(bid, bname, color)

- Assume a Boats relation
- UNION: Can be used to compute the union of any two *union-compatible* sets of tuples
  - can themselves be the result of SQL queries
- 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' OR B.color='green')
```

```
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'
```

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

Sailors (sid, sname, rating, age)
Reserves(sid, bid, day)
Boats(bid, bname, color)

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

Sailors (sid, sname, rating, age)
Reserves(sid, bid, day)
Boats(bid, bname, color)

- 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

```
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')
```

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'
```

Nested Queries

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

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

Sailors (sid, sname, rating, age)
Reserves(sid, bid, day)
Boats(bid, bname, color)

- A very powerful feature of SQL:
  - a WHERE/FROM/HAVING clause can itself contain an SQL query
- 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

Nested Queries with Correlation

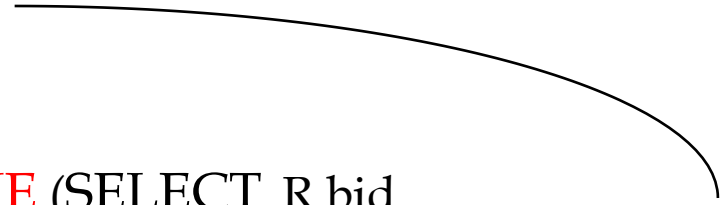
Find names of sailors who've reserved boat #103:

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

- EXISTS is another set comparison operator, like IN
- Illustrates why, in general, subquery must be re-computed for each Sailors tuple

Nested Queries with Correlation

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



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

- 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

More on Set-Comparison Operators

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

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

Aggregate Operators

Check yourself:
What do these queries compute?

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

single column

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

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

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

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

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

Motivation for Grouping

- So far, we've applied aggregate operators to all (qualifying) tuples
  - Sometimes, we want to apply them to each of several groups of tuples
- Consider: Find the age of the youngest sailor for each rating level
  - In general, we don't know how many rating levels exist, and what the rating values for these levels are!
  - Suppose we know that rating values go from 1 to 10; we can write 10 queries that look like this (need to replace i by num):

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

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

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
  - (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
  - Here a group is a set of tuples that have the same value for all attributes in grouping-list

Find age of the youngest sailor with age >= 18, for each rating with at least 2 such sailors.

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

Sailors instance:

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

Answer relation:

rating	minage
3	25.5
7	35.0
8	25.5

Find age of the youngest with age >= 18, for each rating with at least 2 such sailors.

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

Step 1: Form the cross product: FROM clause
(some attributes are omitted for simplicity)

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5

Find age of the youngest with age >= 18, for each rating with at least 2 such sailors.

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

Step 2: Apply WHERE clause

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5

Find age of the youngest with age >= 18, for each rating with at least 2 such sailors.

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

Step 3: Apply GROUP BY according to the listed attributes

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5

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

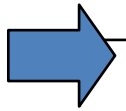
Find age of the youngest with age >= 18, for each rating with at least 2 such sailors.

Step 4: Apply HAVING clause

The *group-qualification* is applied to eliminate some groups

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5



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

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

Find age of the youngest with age >= 18, for each rating with at least 2 such sailors.

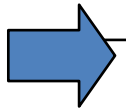
Step 5: Apply SELECT clause

Apply the aggregate operator

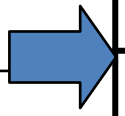
At the end, one tuple per group

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5



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



rating	minage
3	25.5
7	35.0
8	25.5

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

Nulls and Views in SQL

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.

Standard Boolean 2-valued logic

- True = 1, False = 0
- Suppose X = 5
  - (X < 100) AND (X >= 1) is T $\wedge$ T = T
  - (X > 100) OR (X >= 1) is F $\vee$ T = T
  - (X > 100) AND (X >= 1) is F $\wedge$ T = F
  - NOT(X = 5) is ¬T = F

- Intuitively,
  - T = 1, F = 0
  - For V1, V2 $\in$ {1, 0}
  - V1 $\wedge$ V2 = MIN (V1, V2)
  - V1 $\vee$ V2 = MAX(V1, V2)
  - ¬(V1) = 1 – V1

2-valued logic does not work for nulls

- Suppose rating = null, X = 5
- Is rating>8 true or false?
- What about AND, OR and NOT connectives?
  - (rating > 8) AND (X = 5)?
- What if we have such a condition in the WHERE clause?

3-Valued Logic For Null

- TRUE (= 1), FALSE (= 0), UNKNOWN (= 0.5)
  - unknown is treated as 0.5
- Now you can apply rules from 2-valued logic!
  - For V1, V2 $\in$ {1, 0, 0.5}
  - V1 $\wedge$ V2 = MIN (V1, V2)
  - V1 $\vee$ V2 = MAX(V1, V2)
  - ¬(V1) = 1 – V1
- Therefore,
  - NOT UNKNOWN = UNKNOWN
  - UNKNOWN OR TRUE = TRUE
  - UNKNOWN AND TRUE = UNKNOWN
  - UNKNOWN AND FALSE = FALSE
  - UNKNOWN OR FALSE = UNKNOWN

New issues for Null

- The presence of null complicates many issues. E.g.:
  - Special operators needed to check if value IS/IS NOT NULL
  - Be careful!
  - "WHERE X = NULL" does not work!
  - Need to write "WHERE X IS NULL"
- Meaning of constructs must be defined carefully
  - e.g., WHERE clause eliminates rows that don't evaluate to true
  - So not only FALSE, but UNKNOWNs are eliminated too
  - very important to remember!
- But NULL allows new operators (e.g. outer joins)
- Arithmetic with NULL
  - all of +, -, *, / return null if any argument is null
- Can force "no nulls" while creating a table
  - sname char(20) NOT NULL
  - primary key is always not null

Aggregates with NULL

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

R1

- What do you get for
- SELECT count(*) from R1?
- SELECT count(rating) from R1?

Aggregates with NULL

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

R1

- What do you get for
- SELECT count(*) from R1?
- SELECT count(rating) from R1?
- Ans: 3 for both

Aggregates with NULL

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

R1

sid	sname	rating	age
22	dustin	7	45
31	lubber	null	55
58	rusty	10	35

R2

- What do you get for
- SELECT count(*) from R1?
- SELECT count(rating) from R1?
- Ans: 3 for both
- What do you get for
- SELECT count(*) from R2?
- SELECT count(rating) from R2?

Aggregates with NULL

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

R1

sid	sname	rating	age
22	dustin	7	45
31	lubber	null	55
58	rusty	10	35

R2

- What do you get for
- SELECT count(*) from R1?
- SELECT count(rating) from R1?
- Ans: 3 for both
- What do you get for
- SELECT count(*) from R2?
- SELECT count(rating) from R2?
- Ans: First 3, then 2

Aggregates with NULL

- COUNT, SUM, AVG, MIN, MAX (with or without DISTINCT)
  - Discards null values first
  - Then applies the aggregate
  - Except count(*)
- If only applied to null values, the result is null

sid	sname	rating	age
22	dustin	7	45
31	lubber	null	55
58	rusty	10	35

R2

- SELECT sum(rating) from R2?
- Ans: 17

sid	sname	rating	age
22	dustin	null	45
31	lubber	null	55
58	rusty	null	35

R3

- SELECT sum(rating) from R3?
- Ans: null

Views

- A view is just a relation, but we store a definition, rather than a set of tuples

```
CREATE  VIEW YoungActiveStudents (name, grade)
    AS  SELECT   S.name, E.grade
    FROM Students S, Enrolled E
    WHERE S.sid = E.sid and S.age<21
```

- Views can be dropped using the DROP VIEW command

- Views and Security: Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s)
  - the above view hides courses “cid” from E

- More on views later in the course

Can create a new table from a query on other tables too

SELECT… INTO…. FROM…. WHERE

```
SELECT S.name, E.grade
INTO YoungActiveStudents
FROM Students S, Enrolled E
WHERE S.sid = E.sid and S.age<21
```

“WITH” clause – very useful!

- You will find “WITH” clause very useful!

```
WITH Temp1 AS
    (SELECT ….. ..),
    Temp2 AS
    (SELECT ….. ..)
SELECT X, Y
FROM TEMP1, TEMP2
WHERE….
```

- Can simplify complex nested queries

Overview: General Constraints

- Useful when more general ICs than keys are involved
- There are also ASSERTIONS to specify constraints that span across multiple tables
- There are TRIGGERS too : procedure that starts automatically if specified changes occur to the DBMS

```
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)))
```

Only FYI, not covered in detail

Triggers

- Trigger: procedure that starts automatically if specified changes occur to the DBMS
- Three parts:
  - 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
```

Summary

- SQL has a huge number of constructs and possibilities
  - You need to learn and practice it on your own
  - Given a problem, you should be able to write a SQL query and verify whether a given one is correct
- Pay attention to NULLs
- Can limit answers using “LIMIT” or “TOP” clauses
  - e.g. to output TOP 20 results according to an aggregate
  - also can sort using ASC or DESC keywords