Lecture 2

Outline

- More SQL
- Constraints
- SQL Embedding

Reading:
R&G: Ch5

Last Time

Basic Single-Table Queries

- Simplest version is straightforward
  - ✓ Produce all tuples in the table that satisfy the predicate
  - ✓ Output the expressions in the SELECT list
  - ✓ Expression can be a column reference, or an arithmetic expression over column refs

```
SELECT [DISTINCT] <column expression list>
FROM <single table>
[WHERE <predicate>]
[GROUP BY <column list>
[HAVING <predicate>] ]
[ORDER BY <column list>] ;
```


Query Semantics

- FROM : compute cross product of tables
- WHERE : Check conditions, discard tuples that fail
- SELECT : Delete unwanted fields
- DISTINCT (optional) : eliminate duplicate rows
- Note: likely a terribly inefficient strategy!
  - ✓ Query optimizer will find more efficient plans

The SQL DDL


```
CREATE TABLE Sailors (
   sid INTEGER,
   sname CHAR(20),
   rating INTEGER,
   age REAL,
   PRIMARY KEY (sid));
CREATE TABLE Boats (
   bid INTEGER,
   bname CHAR (20),
   color CHAR(10),
   PRIMARY KEY (bid));
CREATE TABLE Reserves (
   sid INTEGER,
   bid INTEGER,
   day DATE,
   PRIMARY KEY (sid, bid, day),
   FOREIGN KEY (sid) REFERENCES Sailors,
   FOREIGN KEY (bid) REFERENCES Boats);
```

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

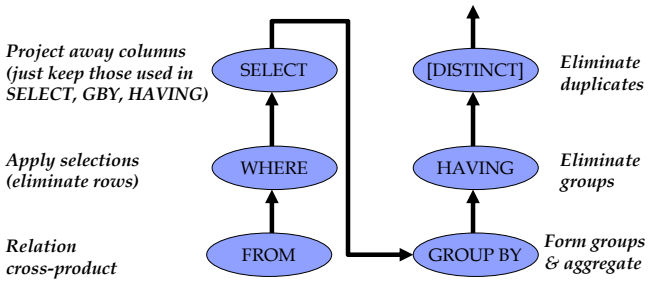
bid	bname	color
101	Nina	red
102	Pinta	blue
103	Santa Maria	red

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

More SQL

Conceptual SQL Evaluation


```
SELECT [DISTINCT] <column expression list>
FROM <single table>
[WHERE <predicate>]
[GROUP BY <column list>
[HAVING <predicate>] ]
[ORDER BY <column list>] ;
```



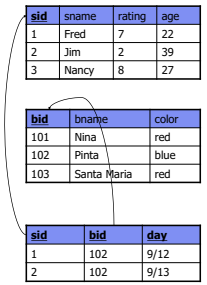
"OR" Example

- 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') ;
```



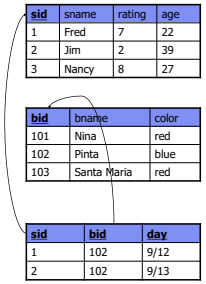
"AND" Example (1)

- 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') ;
```

Which one is correct?

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



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

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

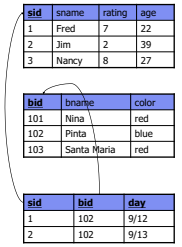
sid	bid	day
1	102	9/12
2	102	9/13

"AND" Example (2)

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

self-join

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



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

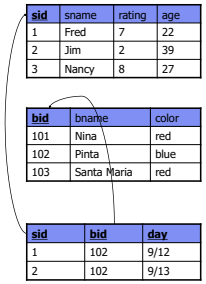
bid	bname	color
101	Nina	red
102	Pinta	blue
103	Santa Maria	red

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

"EXCEPT" Example

- Find sid's of sailors who <u>have not</u> reserved a boat

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



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

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

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

Side Note

- UNION / INTERSECT / EXCEPT reduce duplicates

✓ use UNION ALL / INTERSECT ALL / EXCEPT ALL if you want to consider duplicates

Nested Queries (1)

- EXISTS checks if the set is empty
- Names of sailors who've reserved boat #102:

```
SELECT S.name
FROM Sailors S
WHERE S.sid IN (SELECT R.sid
                FROM Reserves R
                WHERE R.bid = 102);
```

```
SELECT S.name
FROM Sailors S
WHERE EXISTS (SELECT *
              FROM Reserves R
              WHERE R.bid = 102 AND S.sid = R.sid);
```

Subquery must be recomputed for each Sailors tuple.

Nested Queries (2)

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

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


Set Comparison

- Find sailors whose rating is greater than that of <u>some</u> sailor called "Fred":

```
SELECT S.name
FROM Sailors S
WHERE S.rating > ANY (SELECT S2.rating
                      FROM Sailors S2
                      WHERE S2.sname = 'Fred')
;
```


ARGMAX

- The sailor with the highest rating

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

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

```
SELECT S.name
FROM Sailors S
ORDER BY S.rating DESC
LIMIT 1
;
```

Null Values (1)

- Field values are sometimes unknown or inapplicable
  - ✓ SQL provides a special value *NULL* for such situations
- The presence of null complicates many issues
  - ✓ Special syntax “IS NULL” and “IS NOT NULL”

Null Values (2)

- Assume rating = *NULL*. Consider predicate “rating>8”
  - ✓ True? False?
  - ✓ What about AND, OR and NOT connectives?
  - ✓ SUM?
- We need a 3-valued logic (true, false and unknown)
  - ✓ Meaning of constructs must be defined carefully. (e.g., WHERE clause eliminates rows that don’t evaluate to true.) Usually NULL is considered FALSE in WHERE clauses
  - ✓ New operators (in particular, outer joins) possible/needed

Null Logic

- Three-valued logic

NOT	
T	F
F	T
N	N

AND	T	F	N
T	T	F	N
F	F	F	F
N	N	F	N

OR	T	F	N
T	T	T	T
F	T	F	N
N	T	N	N

Views

- View are named queries
  - ✓ Makes development simpler
  - ✓ Often used for security
  - ✓ can be used as relations in queries, similar to subqueries
  - ✓ Not “materialized”

```
CREATE VIEW <view name>
AS <select_statement>
```


```
CREATE VIEW Red_reserves
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
;
```

Subqueries in FROM

Use subqueries:

```
SELECT bname, ncount
FROM Boats B,
     (SELECT B.bid, COUNT(*) AS ncount
      FROM Boats B, Reserves R
      WHERE R.bid = B.bid AND B.color = 'red'
      GROUP BY B.bid
     ) AS Red_reserves(bid, ncount)
WHERE Red_reserves.bid = B.bid AND ncount <10;
```

Use Views:

```
CREATE VIEW Red_reserves
AS
  SELECT B.bid, COUNT(*) AS ncount
  FROM Boats B, Reserves R
  WHERE R.bid = B.bid AND B.color = 'red'
  GROUP BY B.bid
;
```

```
SELECT bname, ncount
FROM Red_reserves R, boats B
WHERE R.bid = B.bid
  AND ncount < 10
;
```


WITH

- WITH clause allows you to give a sub-query block a name
  - ✓ sub-query refactoring
  - ✓ can be referenced in several places *within the main SQL query*
  - ✓ temporary

```
WITH Red_reserves(bid, ncount) AS
     (SELECT B.bid, COUNT(*) AS ncount
      FROM Boats B, Reserves R
      WHERE R.bid = B.bid AND B.color = 'red'
      GROUP BY B.bid
     )
SELECT bname, ncount
FROM Boats B, Red_reserves
WHERE Red_reserves.bid = B.bid AND ncount <10;
```


Discretionary Access Control

- Object can be a Table or a View
- Privileges can be:

```
GRANT <privileges> ON <object> TO <users>
[WITH GRANT OPTION]
```

  - ✓ Select / Insert / Delete
  - ✓ References (cols) – allow to create a foreign key that references the specified column(s)

```
GRANT SELECT ON EMPLOYEE TO THY;
```

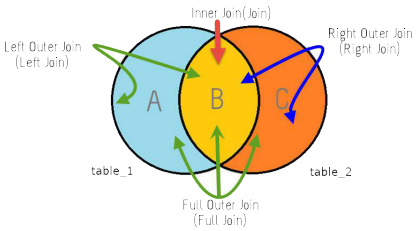
  - ✓ All
- Can later be changed using REVOKE
- Users can be single users or groups

Reading:
R&G: Ch17

JOIN

Join

- INNER JOIN is default




```
SELECT [DISTINCT] <column expression list>
FROM <single table>
[INNER | {LEFT |RIGHT | FULL } {OUTER} ] JOIN <table name>
   ON <qualification list>
[WHERE <predicate>]
[GROUP BY <column list> [HAVING <predicate>] ]
[ORDER BY <column list>] ;
```


Inner/Natural Joins

- "NATURAL" means equi-join for each pair of attributes with the same name
- The following 3 ways are equivalent!

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

```
SELECT S.sid, S.sname, R.bid
FROM Reserves R
INNER JOIN Sailors S
      ON  R.sid = S.sid
;
```

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

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

```
SELECT S.sid, S.sname, R.bid
FROM Reserves R
NATURAL JOIN Sailors S
;
```


Inner/Natural Joins


```
SELECT S.sid, S.sname, R.bid
FROM Reserves R
NATURAL JOIN Sailors S
;
```

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 NULL in fields of non-matching tuples
- Ex: Returns all sailors & bid for the boat in any of their reservations

```
SELECT S.sid, S.sname, R.bid
FROM Sailors S
LFET OUTER JOIN Reserves R
            ON  R.sid = S.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*
- Ex: Returns all boats & information of the sailor who reserve them.

```
SELECT R.sid, B.bid, B.bname
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*
- Ex: Returns all boats & all information on reservations
  - ✓ 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.
-
```
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

Constraints

Integrity Constraints

- IC conditions that every legal row of a relation must satisfy
  - ✓ Inserts/deletes/updates that violate ICs 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: Field values must be of the right type. Always enforced.
  - ✓ Primary key and foreign key constraints
  - ✓ General constraints

Where Do ICs Come From

- Semantics of the real world !
  - ✓ We can check IC violation in a DB instance
  - ✓ We can NEVER infer that an IC is true by looking at an instance.
  - ✓ An IC is a statement about all possible instances!
    - For 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

Key

- Key is a way to associate tuples in different relations
- Key is one form of IC

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

FOREIGN Key

PRIMARY Key

Primary Key (1)

- A set of fields is a superkey if
  - ✓ No two distinct tuples can have same values in all key fields
- A set of fields is a key for a relation if it is a *minimal* superkey
  - ✓ It is a superkey
  - ✓ No subset of the fields is a superkey

R1

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

Boats

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

Primary Key (2)

- What if >1 key for a relation?
  - ✓ One of the keys is chosen (by DBA) to be the primary key
  - ✓ Other keys are called candidate keys
- E.g.
  - ✓ `sid` is a key for `Students`
  - ✓ What about name?
  - ✓ The set {sid, gpa} is a superkey

Primary vs. Candidate Keys

- Possibly many candidate keys (specified using UNIQUE), one of which is chosen as the primary key
- Keys must be used carefully!
  - ✓ (1) "For a given student and course, there is a single grade."
  - ✓ (2) "Students can take only one course, and no two students in a course receive the same grade."

```
CREATE TABLE Enrolled1 (
   sid CHAR(20),
   cid CHAR(20),
   grade CHAR(2),
   PRIMARY KEY (sid, cid));
```

```
CREATE TABLE Enrolled2 (
   sid CHAR(20),
   cid CHAR(20),
   grade CHAR(2),
   PRIMARY KEY (sid),
   UNIQUE (cid,grade));
```


Foreign Key

- Foreign key: a "logical pointer"
  - ✓ Set of fields in a tuple in one relation that 'refer' to a tuple in another relation
  - ✓ Reference to the (***primary) key*** of the other relation
- All foreign key constraints enforced?
  - ✓ referential integrity! (RI)
  - ✓ i.e., no dangling references

Foreign Key

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

```
CREATE TABLE Enrolled1 (
   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
~~11111~~	~~English102~~	~~A~~

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 RI (1)

- `sid` in `Enrolled`: foreign key referencing `Students`
- Scenarios:
  - ✓ Insert Enrolled tuple with non-existent student id?
  - ✓ Delete a Students tuple?

Enforcing RI (2)

- Scenarios:
  - ✓ Delete a Students tuple?
    - Also delete Enrolled tuples that refer to it? (CASCADE)
    - Disallow if referred to? (NO ACTION)
    - Set sid in referring Enrolled tups to a default value? (SET DEFAULT)
    - Set sid in referring Enrolled tuples to null, denoting `unknown' or 'inapplicable' (SET NULL)
- Similar issues arise if the primary key of `Students` tuple is updated.

SQL Embedded in Other Languages

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" — makes "simple" relational language complex
  - ✓ Allow SQL to be embedded in regular programming languages. Q: What needs to be solved?

Using SQL in Apps

- The mismatch between DBMS and programming languages can be solved in several ways
  - ✓ SQL embedding
  - ✓ Cursor
  - ✓ Dynamic SQL
  - ✓ Database API

Cursors (1)

- Can declare a cursor on a relation or query
  - ✓ open a cursor
  - ✓ repeatedly fetch a tuple (moving the cursor)
  - ✓ Special return value when all tuples have been retrieved
- Akin to the concept of “iterator” !!!!
  - ✓ row-by-row actions

Cursors (2)

- ORDER BY allows control over the order tuples are returned
  - ✓ Fields in ORDER BY clause must also appear in SELECT clause
- LIMIT controls the number of rows returned (good fit w/ ORDER BY)
- Can also modify/delete tuple pointed to by a cursor
  - ✓ A “non-relational” way to get a handle to a particular tuple

Database APIs (1)

- 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. in Ruby there’s the “pg” gem for Postgres

Database APIs (2)

- ODBC/JDBC try to be DBMS-neutral
  - ✓ at least try to hide distinctions across different DBMSs
- Object-Relational Mappings (ORMs)
  - ✓ Ruby on Rails, Django, Spring, BackboneORM, etc.
  - ✓ Automagically map database rows into programming objects
  - ✓ Magic can be great; magic can bite you

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

End of Lecture

Review Questions (1)

- Please provide the difference between “UNION” and “UNION ALL”.
- What is a SQL view? Provide possible pros of using views.
- Describe the concept of NULL. How can you check if a field is NULL in SQL?
- What are integrity constrains (ICs)? Please also provide at least three types of ICs.

Review Questions (2)

- Write a SQL query for the following using the provided schema.
  - ✓ Find sid’s of sailors who’ve reserved a red or a green boat
  - ✓ Find sid’s of sailors who’ve reserved a red and a green boat
  - ✓ Find sid’s of sailors who have not reserved a boat
  - ✓ The sailor with the highest rating
  - ✓ Find sailors whose rating is greater than that of some sailor called “Fred"
  - ✓ Names of sailors who’ve not reserved boat #103

Sailors

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

Boats

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

Reservation

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

Review Questions (3)

- Consider the following two relations. Provide the SQL queries and the query result for the following:
  - ✓ INNER JOIN of Sailors on Reserves.
  - ✓ NATURAL JOIN of Sailors on Reserves.
  - ✓ LEFT JOIN JOIN of Sailors on Reserves.
  - ✓ RIGHT JOIN of Sailors on Reserves.
  - ✓ FULL OUTER JOIN of Sailors on Reserves.

Sailors

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

Reserves

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

Review Questions (4)

- Define the following: (1) Super Key (2) Key (3) Primary Key (4) Candidate Key (5) Search Key (6) Index (7) Foreign key
- Describe at least two basic ways to access databases from applications.
- Explain the how the cursors work.