SQL

Introduction

- Structured Query Language (SQL) is the most widely used commercial relational database language.
- It was originally developed at IBM in the SEQUEL-XRM and System-R projects (1974-1977).
- Almost immediately, other vendors introduced DBMS products based on SQL, and it is now a de facto standard.
- The SQL continues to evolve in response to the changing need.

Introduction

- The SQL language has several aspects to it:
  - The Data Manipulation Language (DML)
    - The subset of SQL allows users to pose queries and to insert, delete, and modify rows.
  - The Data Definition Language (DDL)
    - The subset of SQL supports the creation, deletion, and modification to definitions for tables and views.
  - Triggers and Advanced Integrity Constraints
    - The feature was introduced in SQL:1999. The standard includes support for triggers, which are action executed by the DBMS whenever changes to the database meet conditions specified in the trigger

Create Table Statement

- A relational database can store a lot of relations, each relation is created with the ***create table*** statement.
- Example: create a relation that stores info about people.
- CREATE TABLE person(

    drivers_license VARCHAR(20) PRIMARY KEY,

    name VARCHAR(20)

  );
- *Note: Technically,* ***create*** *is the main command, where* ***table*** *is one of the many arguments create takes.*

Drop Table Command

- When a relational database no longer needs a relations, a relation can be deleted from the database using the ***drop table*** statement.
- **DROP TABLE** person;
- *Note: Just like create,* ***drop*** *is the main command, where* ***table*** *is one of the many arguments drop takes.*

Table Attributes

Within the create table command, we specify it's the columns/attributes, declare properties of the table and the properties of each attribute

```
CREATE TABLE table (
        attribute1  datatype [properties],
        attribute2  datatype [properties],
        …
        [table property1]
        [table property2]
        …
);
```

Attribute Constraints

- Attributes properties can be used to “enforce” more complex domain membership conditions.

```
CREATE TABLE table (
        attribute1  datatype [properties],
        attribute2  datatype [properties],
        …
        [table property1]
        [table property2]
        …
);
```

Data Types - Numeric

- Some options for specifying the attributes for holding numeric values:
- If you need integers
  1. ***smallint*** (2 byte integer)
  2. ***int*** (4 byte integer)
  3. ***bigint*** (8 byte integer)
- If you need real numbers
  1. ***real*** (4 byte floating point)
  2. ***double*** (8 byte floating point)
  3. ***numeric (<precision> , <scale>)***
     - *<precision>:* specify significant figures
     - *<scale>*: specify digits after the decimal point

Data Types – String Literal

- Example of a string literal: 'John'
- A string literal is **a sequence of zero or more characters**
- In SQL, you specify a literal by enclosing it in single quotes.
- Two kinds of string literals are available:
  - ***CHAR(n)*** *n length, left-justified blank-padded*
  - ***VARCHAR(n)*** *can be between 0 and n length, no padding*
- String literals are case sensitive: 'John' != 'JOHN '

Attribute Constraints

By default, the NULL value is a member of all data types.

Example:

```
CREATE TABLE Likes (
        drinker VARCHAR(20) NOT NULL
        beer VARCHAR(30)
);
```

More exist: UNIQUE, CHECK, DEFAULT, …

Declaring Primary Keys

- Declare primary key constraint with the table property
  - **primary key** (Ai, ...,)

Example: declare the name of a person to be primary key

```
CREATE TABLE Person (
        name VARCHAR(20),
        PRIMARY KEY (name)
);
```

Primary key declaration on an attribute ensures **not null** and **unique**

Note: there an equivalent syntax as an attribute property

Declaring Foreign Keys

- Declare foreign keys with the foreign key constraint
  - **foreign key** (Aj, ...,) **references** r

Example: declare the name of a person to be primary key

```
CREATE TABLE employee (
        name VARCHAR(20)
        works_at VARCHAR(20)
        FOREIGN KEY (works_at )  REFERENCES company(id)
);
```

Note: there an equivalent syntax as an attribute property

Recall: Declaring Foreign Keys

- Declaring foreign keys assures referential integrity.
- Foreign a key:
  - specify Relation (Attribute) to which it refers.
- For instance, if we want to delete a tuple from Beers, and there are tuples in Sells that refer to it, we could either:
  - reject the deletion
  - cascade the deletion and remove Sells records
  - set-NULL the foreign key attribute
- Can force cascade via ON DELETE CASCADE after REFERENCES.

Declaring Foreign Keys

- The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.
- If you want prefer to delete all records referring to the foreign key value in the database, you can specify ON DELETE CASCADE

```
CREATE TABLE Person (
   name VARCHAR(20)
   ateburger VARCHAR(20)
   FOREIGN KEY(ateburger)  REFERENCES burgers (Bid) ON
DELETE CASCADE
);
```

Basic Query Structure

- To retrieve information from a database, there is a basic query structure, known as the ***select*** statement.
- SELECT <Attribute list>
  FROM <Table list>
  WHERE <Condition>
- <attribute list>: list of attributes
  <table list>: list of relations
  <condition>: list of conditions (Boolean expression)
- SELECT statement is also known as a ***select-from-where block***.

The Select Clause

- The **select** clause lists the attributes desired in the result of a query
  - corresponds to the projection operation of the relational algebra

Example: Give all the names of all drinkers

```
SELECT Name
FROM Drinkers;
```

Drinkers:

Name	Addr	Phone
Adam	Randwick	9385-4444
Gernot	Newtown	9415-3378
John	Clovelly	9665-1234
Justin	Mosman	9845-4321

- *Note: FROM is always necessary with SELECT, whereas WHERE is optional.*

The Select Clause

- Example: Give me both names and addresses of drinkers!
- SELECT **Name, Addr**
  FROM Drinkers;

Drinkers:

Name	Addr	Phone
Adam	Randwick	9385-4444
Gernot	Newtown	9415-3378
John	Clovelly	9665-1234
Justin	Mosman	9845-4321

- An asterisk in the select clause denotes “all attributes”
- SELECT *
  FROM Drinkers;

Drinkers:

Name	Addr	Phone
Adam	Randwick	9385-4444
Gernot	Newtown	9415-3378
John	Clovelly	9665-1234
Justin	Mosman	9845-4321

The Select Clause

- To eliminate duplicates in the query results, insert the keyword ***distinct*** after select.
- Example: Find the names of all departments and remove duplicates.
- **Select *distinct*** dept_name from instructor
- The keyword **all** specifies that duplicates not to be removed.
- **Select *all*** dept_name from instructor

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table

The Select Clause

- SQL allows duplicates in relations and in query results.
  - allows a table to have two or more tuples that are identical in all their attribute values.
- In general, an SQL table can be a simple set of tuples, or a multiset of tuples.
- Hold on… wasn't SQL a faithful mapping of RM/RA?
- Set: {a, b, c}
  Multiset: {a, a, b, b, c, a, a, b, c, c …}

The From Clause

- The from clause lists the relations involved in the query
  - Corresponds to the ***Cartesian product*** operation of the relational algebra.
- Find the Cartesian product *instructor* × *teaches*

  SELECT *
  FROM *instructor, teaches*

  - generates every possible instructor – teaches pair, with all attributes from both relations.
- *Cartesian product not very useful directly, but useful combined with where-clause condition (selection operation in relational algebra)*.

Cartesian Product

instructor

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000

teaches

ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009

Inst.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci	65000	10101	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci	65000	10101	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci	65000	10101	CS-347	1	Fall	2009
10101	Srinivasan	Comp. Sci	65000	12121	FIN-201	1	Spring	2010
10101	Srinivasan	Comp. Sci	65000	15151	MU-199	1	Spring	2010
10101	Srinivasan	Comp. Sci	65000	22222	PHY-101	1	Fall	2009
…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2009
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2010
12121	Wu	Finance	90000	10101	CS-347	1	Fall	2009
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2010
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2010
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2009
…	…	…	…	…	…	…	…	…

Joins (1)

- For all instructors who have taught courses, find their names and the course ID of the courses they taught.

  SELECT *name, course_id*
  FROM *instructor, teaches*
  WHERE *instructor.ID = teaches.ID*

instructor

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000

teaches

ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009

Joins (2)

- Find instructor names and the courses they taught in 2010.

  SELECT *name, course_id*
  FROM *instructor, teaches*
  WHERE *instructor.ID = teaches.ID* **AND** *year* = 2010

instructor

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000

teaches

ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009

Natural Join (1)

- Natural join matches tuples with the same values for all common attributes, and retains only one copy of each common column
- **SELECT * FROM** instructor **NATURAL JOIN** teaches;

instructor

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000

teaches

ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009

ID	*name*	*dept_name*	*salary*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010

Natural Join (2)

- List the names of instructors along with the titles of courses that they teach. This is an incorrect version:
  - **SELECT** name, title
    FROM instructor
    NATURAL JOIN teaches
    NATURAL JOIN course;

instructor

ID	name	dept_name	salary
8	ABC	SEEM	100
7	XYZ	SEEM	120

teaches

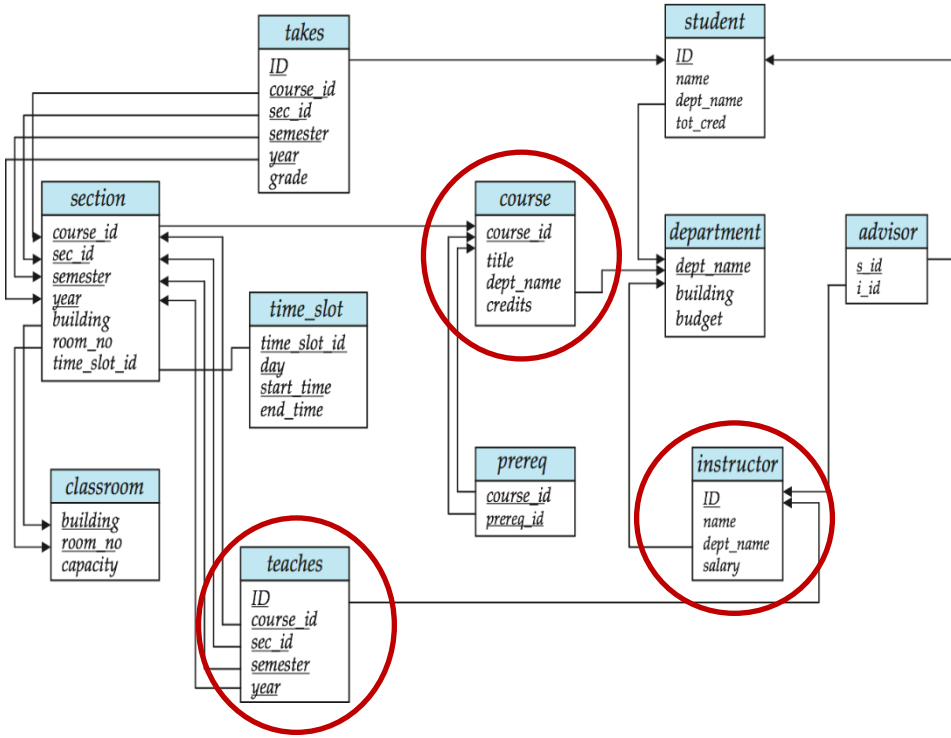
ID	course_id	sec_id	semester	year
7	3550	1	1	2018
8	2100	1	2	2018

course

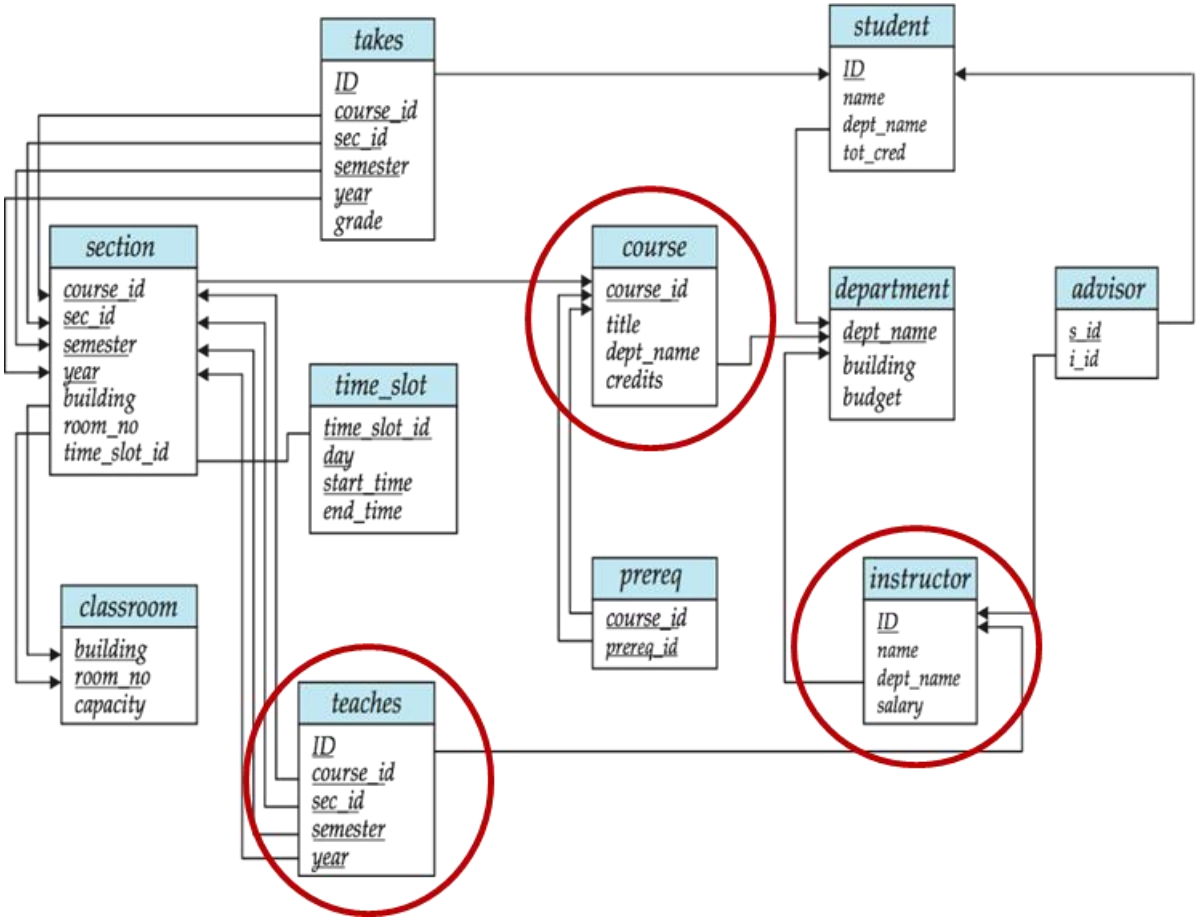
course_id	title	dept_name	credits
3550	DB	SEEM	3
2100	Algo	CSE	3

Natural Join (3)

- Dangers of natural join:
- When you join based on attributes
  with same name but are unrelated
- Example: List the names of **instructors** along with the titles of **courses** that they teach
- Why was this incorrect?
  - **SELECT** name, title
    FROM instructor
    NATURAL JOIN instructor
    NATURAL JOIN course;



Natural Join (3)



1. *Course.dept_name* and *instructor.dept_name* are not related
2. Therefore, cannot be assumed to be the same.

Natural Join (4)

- List the names of instructors along with the titles of courses that they teach. This is a correct version:
- **SELECT** name, title
  FROM instructor
  NATURAL JOIN teaches, course
  WHERE teaches.course_id = course.course_id;

instructor

ID	name	dept_name	salary
8	ABC	SEEM	100
7	XYZ	SEEM	120

teaches

ID	course_id	sec_id	semester	year
7	3550	1	1	2018
8	2100	1	2	2018

course

course_id	title	dept_name	credits
3550	DB	SEEM	3
2100	Algo	CSE	3

Natural Join (4)

- List the names of instructors along with the titles of courses that they teach. This is another correct version:
- **SELECT** name, title
  FROM instructor
  INNER JOIN teaches **ON** instructor.id = teaches.course_id
  INNER JOIN course **ON** instructor.dept_name = course.dept_name
  AND teaches.course_id= course.course_id;

instructor

ID	name	dept_name	salary
8	ABC	SEEM	100
7	XYZ	SEEM	120

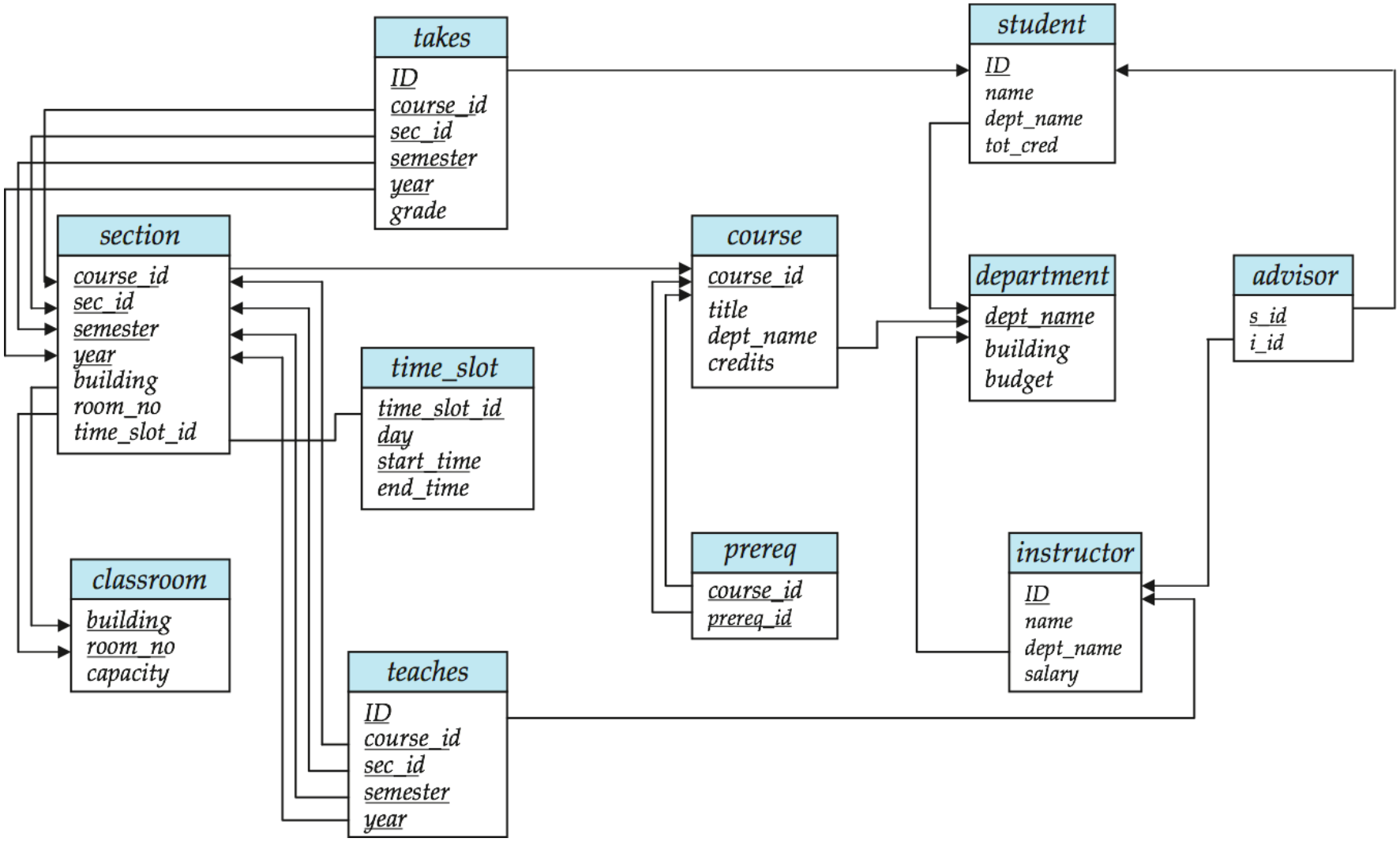
teaches

ID	course_id	sec_id	semester	year
7	3550	1	1	2018
8	2100	1	2	2018

course

course_id	title	dept_name	credits
3550	DB	SEEM	3
2100	Algo	CSE	3

Schema Diagram for University Database



The Where Clause

- The **WHERE** clause specifies conditions that the result must satisfy
  - corresponds to the *selection condition* of the relational algebra.
- To find all bars that sell the beer New
- **SELECT** *
  FROM Sells
  WHERE Beer = 'New';

Sells:

Bar	Beer	Price
Australia Hotel	Burragorang Bock	3.5
Regent Hotel	New	2.2
Regent Hotel	Victoria Bitter	2.2

The Where Clause

- Find the beers manufactured by Toohey's
- **SELECT** Name
  FROM Beers
  WHERE Manf = 'Toohey''s';

Beers:

Name	Manf
80/-	Caledonian
Premium Lager	Cascade
Red	Toohey's
Sheaf Stout	Toohey's
Sparkling Ale	Cooper's
Victoria Bitter	Carlton

- What if my string has single quotes in it?
- Escaping by doubling them ('') to inform SQL that this is part of the string literal.
  - Without escaping
    SELECT Name FROM Beers WHERE Manf = 'Toohey's';) – ERROR

Operational semantics: Select

- 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*.)

Operational semantics: Select

- Example: Find the names of sailors who have reserved boat number 103.

sid	bid	day
22	101	10/10/19
58	103	11/12/19

Instance R3 of Reserves

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

Instance S4 of Sailors

sid	bid	day
22	101	10/10/19
58	103	11/12/19

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

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

S.sid	sname	rating	age	R.sid	bid	day
22	dustin	7	45.0	22	101	10/10/19
22	dustin	7	45.0	58	103	11/12/19
31	lubber	8	55.5	22	101	10/10/19
31	lubber	8	55.5	58	103	11/12/19
58	rusty	10	35.0	22	101	10/10/19
58	rusty	10	35.0	58	103	11/12/19

Row remains after selection.

Result

S4 X R3

Operational semantics: Select

- Operationally, we think in terms of a tuple variable ranging over all tuples of the relation.
- **Operational semantics of SQL SELECT**

```
FOR EACH tuple T in R DO
        check whether T satisfies the condition in the WHERE clause
        IF it does THEN
                print the attributes of T that are
                specified in the SELECT clause
        END
END
```

Ordering Result Tuples

- List in alphabetic order the names of all instructors
  - **SELECT DISTINCT** *name*
    FROM *instructor*
    ORDER BY *name*
- We may specify **DESC** for descending order or **ASC** for ascending order. *The default is ascending order.*
  - Example: … **ORDER BY** *name* **DESC**
- Can sort on multiple attributes
  - Example: … **ORDER BY** *dept_name, name*
- *Didn't we say that relations are unordered?*

Set Operations

- Find courses that ran in Fall 2009 or in Spring 2010

  (**Select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2009)
  union
  (**Select** *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2010)

- Find courses that ran in Fall 2009 and in Spring 2010

  (**Select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2009)
  intersect
  (**Select** *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2010)

- Find courses that ran in Fall 2009 but not in Spring 2010

  (**select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2009)
  except
  (**select** *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2010)

- *Note: Each of the above operations will eliminate duplicates. To keep duplicates, use* ***union all, intersect all, except all.***

Summary of SQL

And Examples

SQL Summary

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

- *SELECT* clause
  - specifies columns to be retained in the result.
- *FROM* clause
  - specifies a cross-product of tables.
- *WHERE* clause (optional)
  - specifies selection conditions on the tables mentioned in the FROM clause.

SELECT DISTINCT $a_1, a_2, \ldots, a_n$

FROM $R_1, R_2, \ldots, R_m$

WHERE P

An SQL query intuitively corresponds to a relational algebra expression involving selections, projections, and cross-products.

$$\Pi_{a1, a2, \ldots, an} (\sigma_P (R_1 \times R_2 \times \ldots \times R_m))$$

Example A

- Example: Find the names of all branches in the loan relation.

```
SELECT branch-name
FROM Loan
```

Loan

branch_name	loan_number	amount
CUHK	222	2
CMTR	333	1
CUHK	777	2

Result

branch_name
CUHK
CMTR
CUHK

Example B

- To remove duplications

```
SELECT DISTINCT branch-name
FROM Loan
```

Loan

branch_name	loan_number	amount
CUHK	222	2
CMTR	333	1
CUHK	777	2

Result

branch_name
CUHK
CMTR

Range Variables

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

- A Note on Range Variables
  - Really needed only if the same relation appears twice in the FROM clause. 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
```

Example C (1)

- Given the following schema:

Sailors(*sid*: integer, *sname*: string, *rating*:integer, *age*: real)

Boats(*bid*: integer, *bname*; string, *color*: string)

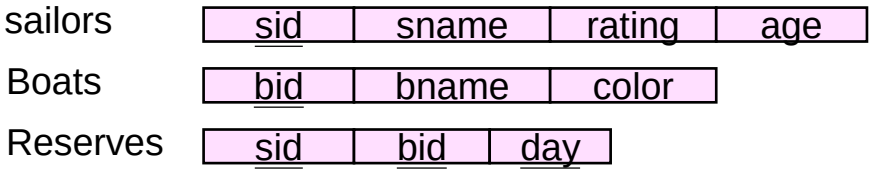
Reserves(*sid*: integer, *bid*: integer, *day*: date)

Sailors	sid	sname	rating	age

Boats	bid	bname	color

Reserves	sid	bid	day

Example C (2)



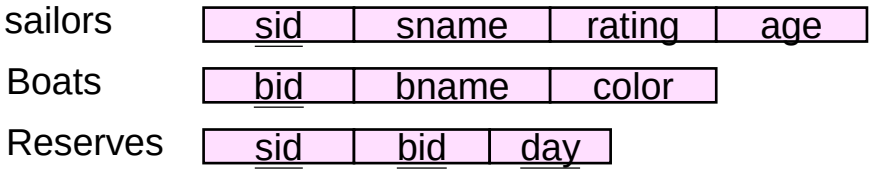
Example1: *Find the sids of sailors who have reserved a red boat.*

```
SELECT  R.sid
FROM  Boat B, Reserves R
WHERE  B.bid = R.bid AND B.color = 'red'
```

Example2: *Find the names of sailors who have reserved a red boat.*

```
SELECT  S.sname
FROM  Sailors S, Reserves R, Boat B
WHERE  S.sid = R.sid AND R.bid = B.bid AND
            B.color = 'red'
```

Example C (3)

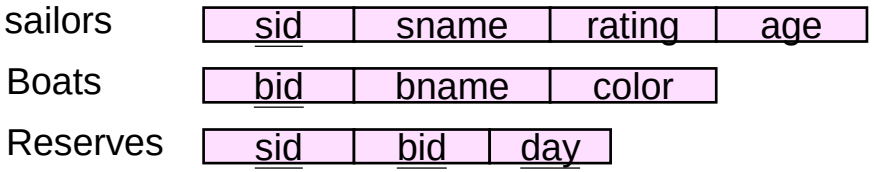


Example3: *Find the colors of boats reserved by Lubber.*

```
SELECT  B.color
FROM  Sailors S, Reserves R, Boat B
WHERE  S.sid = R.sid AND R.bid = B.bid AND
                S.sname = 'Lubber'.
```

(In general, there may be more than one sailor called Lubber. In this case, it will return the colors of boats reserved by all sailors called Lubber).

Example C (4)



Example4: *Find the names of sailors who have reserved at least one boat.*

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

(If a sailor has not made a reservation, the second step in the conceptual evaluation strategy would eliminate all rows in the cross-product that involve this sailor).

Extra: 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.

Appendix: SQL

More on Database Modification

Insert Tuple(s) using values

- The **INSERT** command inserts new tuple(s) into a table
  - **INSERT INTO** Relation **VALUES** (val1, val2, val3, ...)
    (val1, val2, val3, ...) is a tuple of values
- Example: Add the fact that Justin likes 'Old'.
  - INSERT INTO Likes VALUES ('Justin', 'Old');
- The following are the same
  - INSERT INTO Sells (price,bar) VALUES (2.50, 'Coogee Bay Hotel');
    INSERT INTO Sells (bar,price) VALUES ('Coogee Bay Hotel', 2.50);
- *Note: The order of the attributes in VALUES can be different from the SQL table definition as long as the order is specified in the INTO clause*.

Insert Tuple(s) using values

- **Basic attribute constraint checking by SQL**
- E.g., suppose the table specified that drinkers' phone numbers cannot be NULL.
  - **ALTER TABLE** Drinkers **ALTER COLUMN** phone SET NOT NULL;
- And then try to insert a new drinker whose phone number we don't know:
  - **INSERT INTO** Drinkers(name,addr)
    VALUES ('Zoe', 'Manly');
  - ERROR: null value in column "phone" violates not-null constraint
    DETAIL: Failing row contains (Zoe, Manly, null).

Insert Tuple(s) using Select

- We can use the result of a query to perform insertion of multiple tuples at once.
  - **INSERT INTO** Relation (Subquery);
- Condition: Tuples of subquery **must be projected into a suitable format** (by matching the tuple-type of Relation).
- A ***subquery:***
  - a query that is nested inside an outer query: SELECT , INSERT , UPDATE , or DELETE statement.
- Subqueries:
  - can also be nested inside another subqueries.
  - are very helpful.

Insert Tuple(s) using Select

- For Example: Populate a relation of John's potential drinking buddies (i.e. people who go to the same bars as John).

```
CREATE TABLE DrinkingBuddies (
        name varchar(20)
);
INSERT INTO DrinkingBuddies(
        SELECT DISTINCT f2.drinker
        FROM Frequents f1, Frequents f2
        WHERE f1.drinker = 'John'
                AND f2.drinker != 'John'
                AND f1.bar = f2.bar
);
```

Delete Tuple(s)

- The DELETE operation removes all tuples from Table that satisfy a condition.
  - **DELETE FROM** Table
    WHERE Condition
- Example: Justin no longer likes Sparkling Ale.
  - **DELETE FROM** Likes
    WHERE drinker = 'Justin'
    AND beer = 'Sparkling Ale';
- Omitting the WHERE Clause deletes all tuples from relation R.
  - **DELETE FROM** R;
  - This doesn't drop the table, the table still remains
- *Note: Delete is not the same as Drop*

Delete Tuple(s)

- Semantics of the above Deletion:
- Evaluation of DELETE FROM R WHERE Cond can be viewed as:

```
FOR EACH tuple T in R DO
        IF T satisfies Cond THEN
            make a note of this T
        END
END

FOR EACH noted tuple T DO
        remove T from relation R
END
```

Update Value(s) in Tuple(s)

- If you have an tuple but want to change part of its values, use the UPDATE command.
- Updates specified attributes in specified tuples of a relation:
- **UPDATE** R
  SET list of assignments
  WHERE Condition
- Example: John moves to Coogee.
- **UPDATE** Drinkers
  SET addr = 'Coogee' , phone = '9665-4321'
  WHERE name = 'John';
- Note: Careful because all tuples relation R that satisfies Condition has the assignments applied to it.

Update Value(s) in Tuple(s)

- Can update many tuples at once (all tuples that satisfy condition)
- Example: Make $3 the maximum price for beer.
  - **UPDATE** Sells
    SET price = 3.00
    WHERE price > 3.00;
- Example: Increase all beer prices by 10%.
  - **UPDATE** Sells
    SET price = price * 1.10;