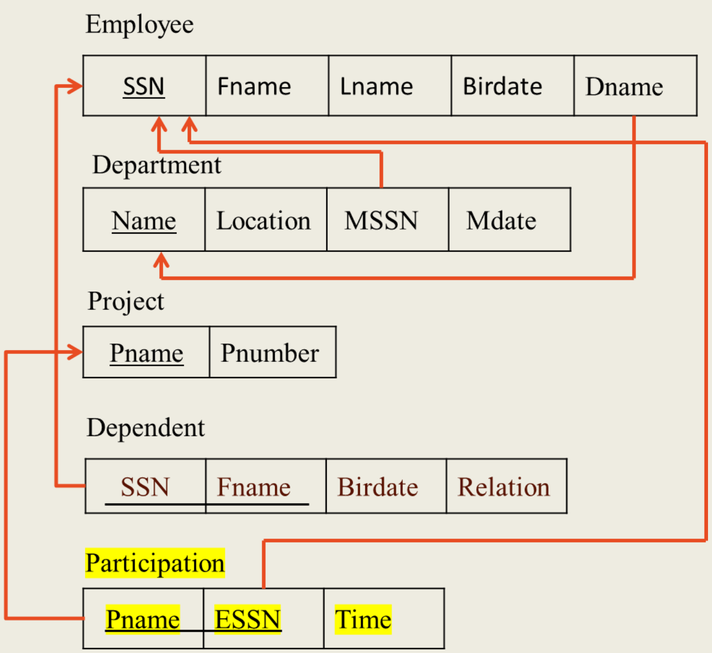
SQL

(Chapters 4,5)

Summary Wk2 Thur

When converting M:N relationship to its own relation R, **why is the final key of R the keys of the two participating entities**?



Summary Wk2 Thur

The ***Precedence*** of relational operators

When you combine operators, use parentheses to emphasize the sequence of

Precedence rules (from highest to lowest)

1. ()
2. [σ, π, ρ]
3. X (order of evaluation left to right)
4. ∩ (order of evaluation left to right)
5. [∪, —] (order of evaluation left to right)

For comparison: the order of operations in

1. ()
2. Exponents
3. [X /] (from left to right)
4. [+ -] (from left to right)

Clarification Wk2 Mon

A clarification on wk2 Monday Slides on the phrasing on n-ary relationships mappings. (Wk2 Monday)

Mapping N-ary Relationship Types

Step 7 : For each ***n-ary relationship type*** (n > 2), create a new relation with

- Attributes : same for Step 5.
- Key :
  - same for Step 5, see exception below
  - The exception is that that if one of the participating entity types has participation ratio 1, its key can used as a key for the new relation.
    - Clarification: it is optional for the attributes from the key for the entity with a cardinality 1 to be part of the key for the new relation

(Advice: binary relationships simpler to model)

Summary Wk2 Thur

What are the columns that remain after a natural join and an equi-join.

SQL-99

1. SQL = ***Structured Query Language*** (pronounced “sequel”).

2. Developed at IBM (San Jose Lab) during the 1970’s, and standardised during the 1980’s.

3. An ANSI/ISO standard language for ***querying and manipulating relational DBMSs***.

4. Was chosen over it’s competing query language QEUL as standard.

SQL in Relational DBMS

In relational databases, what does SQL do?

SQL — A *data manipulation language (DML)*

- used to ***query*** and ***modify*** database data

SQL defines a set of data query commands

- *- SELECT, (keywords relating to select: GROUP BY, HAVING, ORDER BY…)*

SQL defines a set of data manipulation commands

- *- CREATE, DROP, DELETE, UPDATE, ALTER…*

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.*

Objects need Identifiers

Tables in SQL have an ***identifier*** to identify them.

- Letters (defined in the Unicode Standard 3.2.)
- Decimal numbers (must not be first character)
- Whether you can use sign (@), dollar sign ($), number sign (#), or underscore (_) depends on the standard.

The identifier must not be a reserved word/keyword!

Names are not case-sensitive (the same for SQL keywords).

Other SQL objects (e.g., ***attributes, views, …*** *) all have identifiers that follow the above rules.*

Table Attributes - Domain

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]
    …
);
```

Table Attributes - Domain

The syntax requires that attribute in an SQL table must specify a domain.

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

The domain puts constraints on the value that an attribute is allowed to take. Defining the ***data type*** *specifies the domain.*

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 '

Converting Data Types

Conversions between data types are an important skill to know.

E.g., division of one integer with an integer

Various type conversions are available:

- integer to real ...
- string to integer …

SQL supports a small set of useful built-in data types:
e.g., numbers, strings, dates...

- You can define your own type in SQL.

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]
        …
);
```

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

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 a value of a primary key that is referred to in other tables as a foreign key, you risk violating the referential integrity. There are several ways to handle this.

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
[illegible]	[illegible]	[illegible]	[illegible]

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
[illegible]	[illegible]	[illegible]	[illegible]

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
[illegible]	[illegible]	[illegible]	[illegible]

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
[illegible]	[illegible]	[illegible]	[illegible]

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 (3)

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 (2)

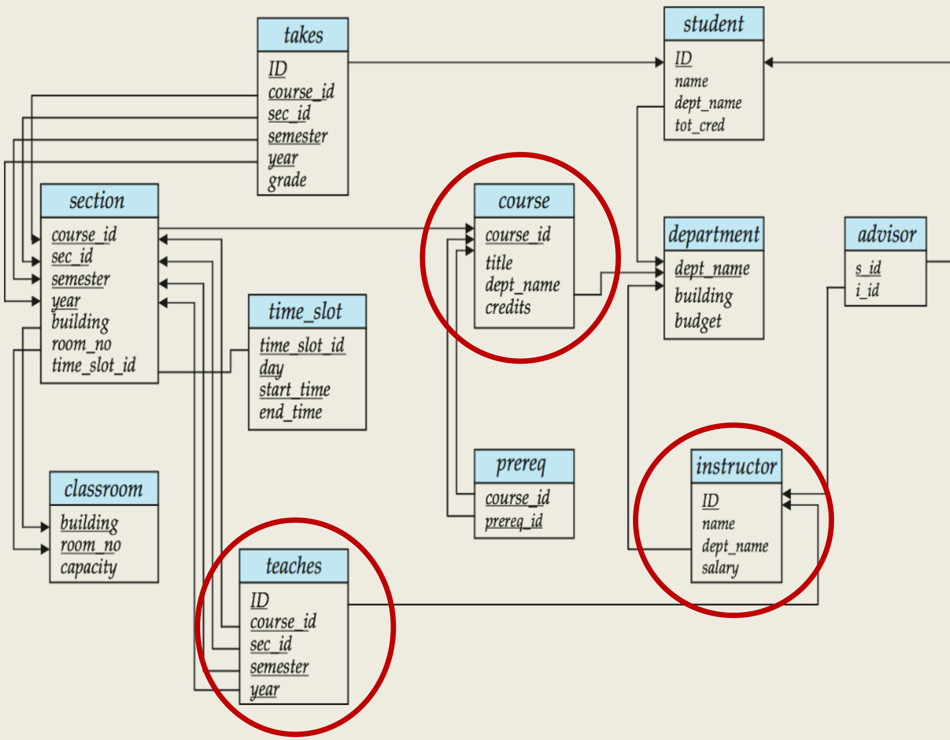
Dangers of natural join:

When you join based on attributes with same name but are unrelated

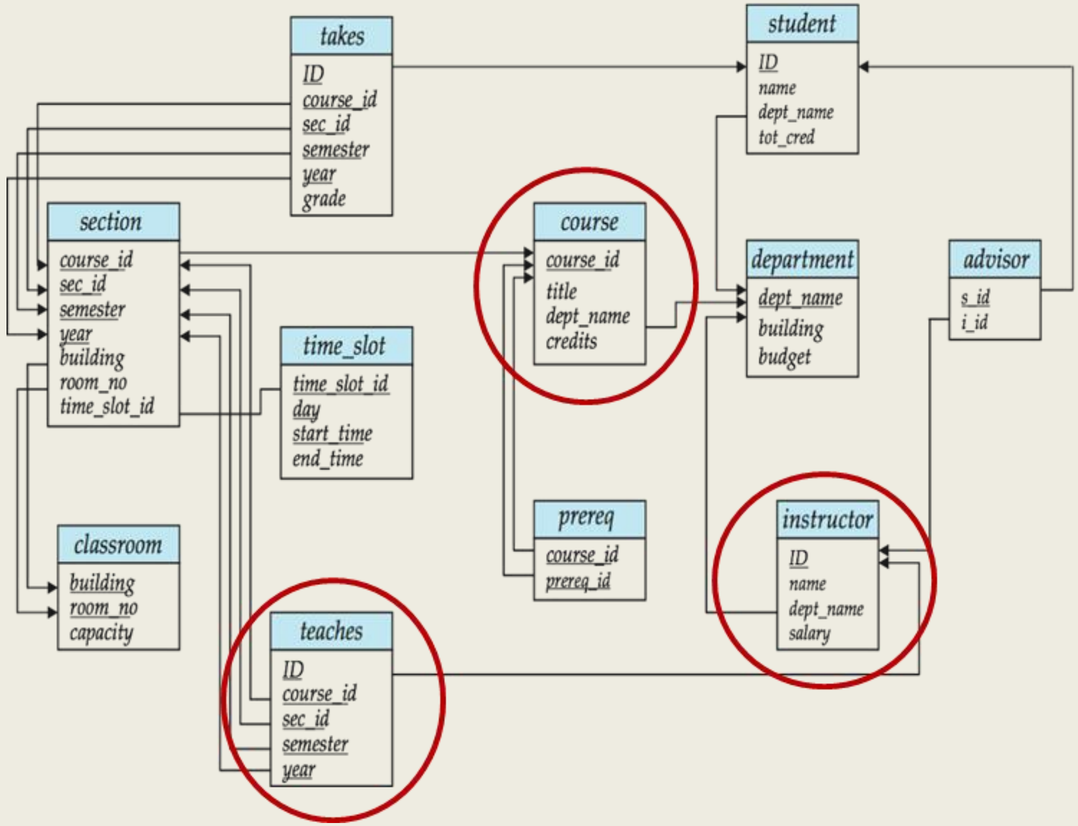
Example: List the names of **instructors** along with the titles of **courses** that they teach

Why was this incorrect?

- **SELECT** name, title
  FROM instructor
  NATURAL JOIN instructor
  NATURAL JOIN course;



Natural Join (2)



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

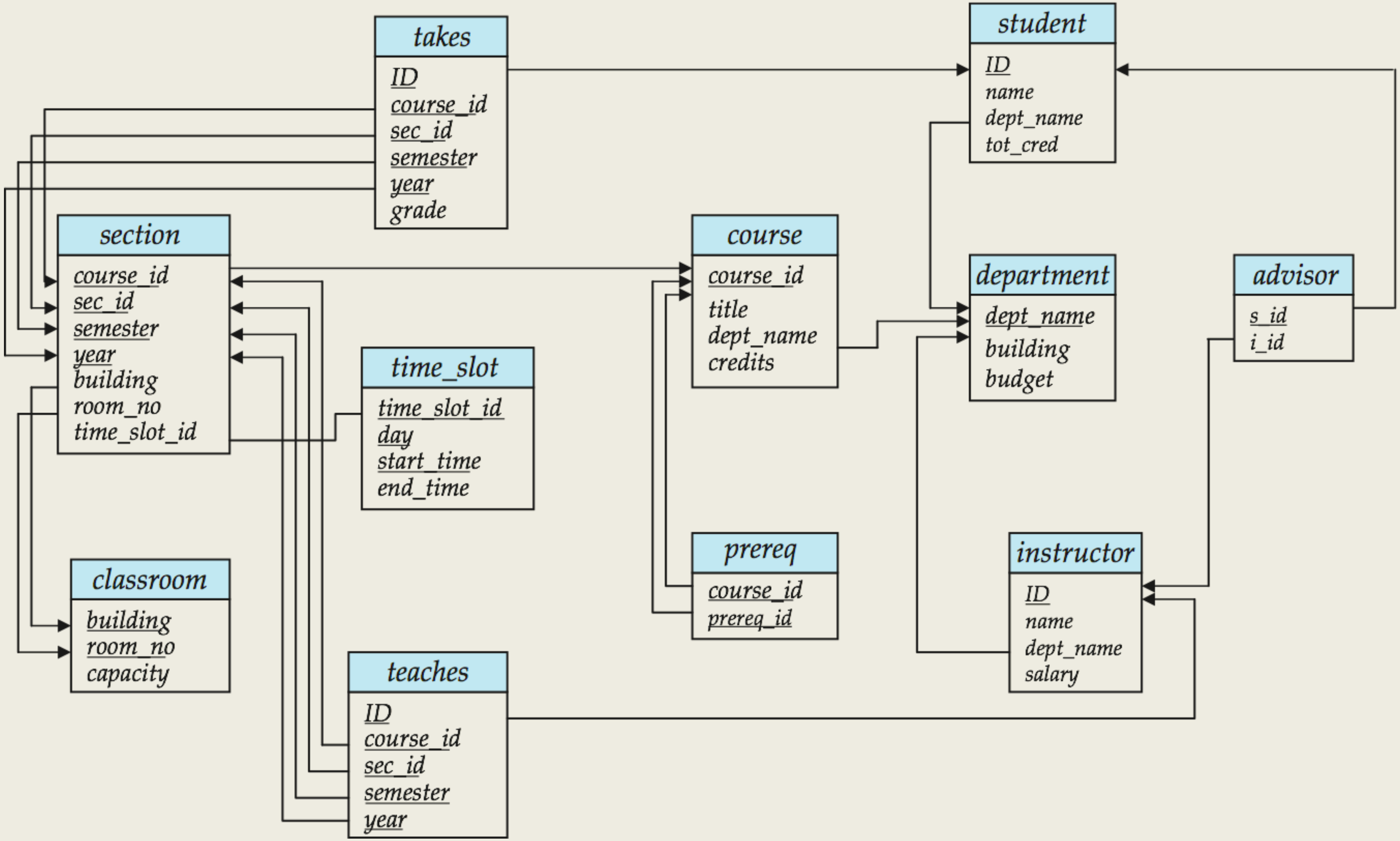
teaches

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

Example Schema Diagram



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?

You do so by escaped 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

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.***

String Operators

1. string **||** string ... concatenate two strings
2. *LENGTH (string)* ... return length of string
3. *SUBSTR (string, start, length)* ... extract chars from within string

Example:

- ‘Post’|| ‘greSQL’ -> PostgreSQL
- substring('Thomas' from 2 for 3) -> hom

SQL Like Operator

str LIKE pattern ... matches string to pattern

Two kinds of string ***pattern-matching***:

- The symbol _ (underscore) matches any single characters
- The symbol % (percent) matches zero or more characters

Practice:

- String LIKE ‘Ja%’ Strings beginning with ‘Ja’
- String LIKE ‘_i%’ Strings with ‘i’ as 2nd letter
- String LIKE ‘%o%o%’ Strings contains two ‘o’s

Null and Three Valued Logic

What happens when the condition makes a comparison with a null value?

Comparisons with null returns unknown

- Example: 5 < null, null <> null, null = null

Three-valued logic using the truth value unknown:

- **OR**: (unknown or true) = true,
  (unknown or false) = unknown,
  (unknown or unknown) = unknown
- **AND**: (true and unknown) = unknown,
  (false and unknown) = false,
  (unknown and unknown) = unknown
- **NOT**: (not unknown) = unknown
- "P is unknown" evaluates to true if predicate P evaluates as unknown

Result of where clause predicate is treated as false if it evaluates as unknown

Example: Null Values

SELECT A3
FROM R
WHERE A1 + 5 > A2 and A4 = ‘x’

A1	A2	A3	A4
5	9	alpha	x
	4	beta	
2	4	gamma	
3		delta	x

When it evaluates the second tuple:

- Null + 5 → Null (for A1 + 5)
- Null > 4 → Null (for A1 + 5 > A2)
- Null = ‘x’ → Null (for A4 = ‘x’)
- Null and Null → Null (for A1 + 5 > A2 and A4 = ‘x’)
- Where clause results false since it is Null. So it does not output “beta”

- What about the following?
  select A3
  from R
  where (A1 + 5 > A2 and A4 = ‘x’) is unknown

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;