CIS 353 Database

SQL

SQL

- SQL = Structured Query Language
- Original language was “SEQUEL”
  - IBM’s System R project (early 1970’s)
  - “Structured English Query Language”
  - Simple, declarative language for writing queries
  - Also includes many other features

SQL Features

- Data Definition Language (DDL)
  - Statements are used to define/modify data structures.
  - For example: create table, alter table, drop SQL statements.
  - Specify relation schemas (attributes, domains)
  - Specify a variety of integrity constraints

SQL Features

- Data Manipulation Language (DML)
  - Statements are used to manipulate data itself
  - Generally based on relational algebra
  - Supports querying, inserting, updating, deleting data
  - Supports features for multi-table queries

SQL Basics

- SQL language is case-insensitive
  - both keywords and identifiers (for the most part)
- SQL statements end with a semicolon
- SQL comments have two forms:
  - Single-line comments start with two dashes -- This is a SQL comment.
  - Block comments follow C style:

```
/*
 * This is a block comment in SQL.
 */
```

DATA Definition LANGUAGE (DDL)

Creating (Declaring) a Relation/Table

- To create a relation:
  CREATE TABLE <name>(
  <list of elements>
  );

- To delete a relation:
  DROP TABLE <name>;
- To alter a relation (add/remove column):
  - ALTER TABLE <name> ADD <element>
  - ALTER TABLE <name> DROP <element>

Creating a SQL Table

- Syntax:

```
CREATE TABLE t (
    attr1 domain1,
    attr2 domain2,
    … ,
    attrN domainN
);
```

- t is name of relation (table)
- attr1, … are names of attributes (columns)
- domain1, … are domains (types) of attributes

Creating a SQL Table: Practice

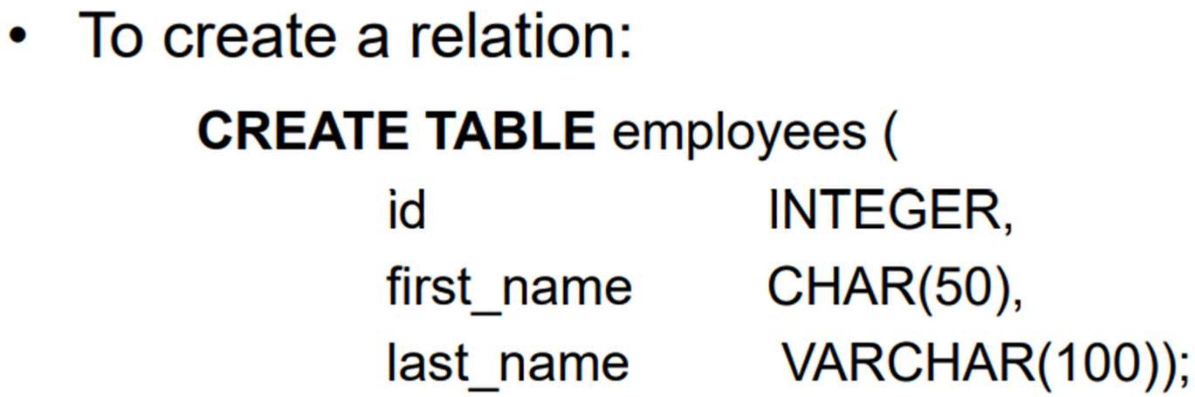
- Syntax:

```
CREATE TABLE t (
    attr1 domain1,
    attr2 domain2,
    … ,
    attrN domainN
);
```

https://livesql.oracle.com/

Creating (Declaring) a Relation/Table

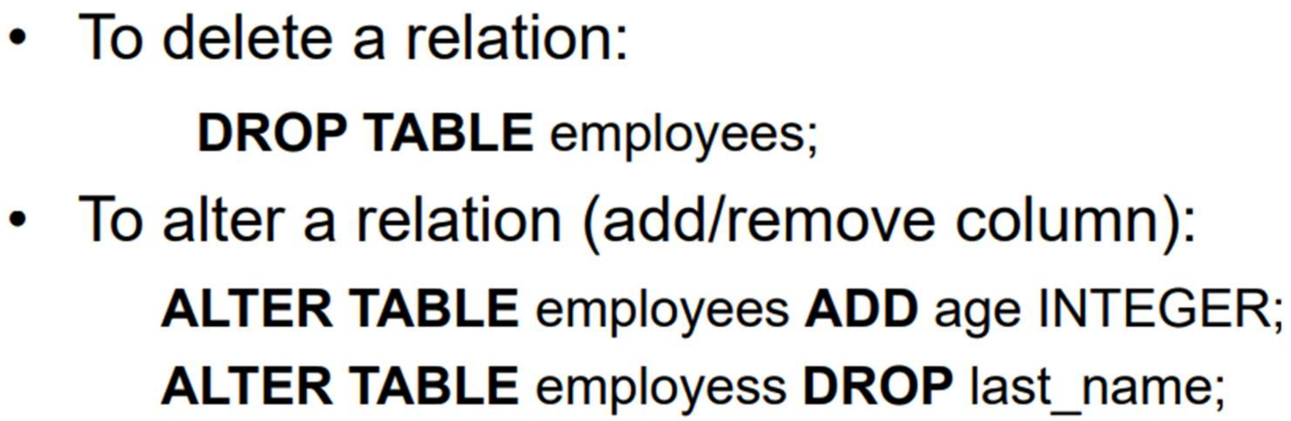
https://livesql.oracle.com/



- To create a relation:

```
CREATE TABLE employees (
        id              INTEGER,
        first_name      CHAR(50),
        last_name        VARCHAR(100));
```

Creating (Declaring) a Relation/Table



- To delete a relation:

```
DROP TABLE employees;
```

- To alter a relation (add/remove column):

```
ALTER TABLE employees ADD age INTEGER;
ALTER TABLE employess DROP last_name;
```

SQL Attribute Domains

- **CHAR(N)**
  - A character field, fixed at N characters wide
  - Short for CHARACTER(N)
- **VARCHAR(N)**
  - A variable-width character field, with maximum length N
  - Short for CHARACTER VARYING(N)
- **INT**
  - A signed integer field (typically 32 bits)
  - Short for INTEGER
  - Also TINYINT (8 bits), SMALLINT (16 bits), BIGINT (64 bits)

SQL Attribute Domains

- **NUMERIC(P,D)**
  - A fixed-point number with user-specified precision
  - P total digits; D digits to right of decimal place
- **DOUBLE PRECISION**
  - A double-precision floating-point value
- **FLOAT(N)**
  - A floating-point value with at least N bits of precision
- **DATE, TIME, TIMESTAMP**
  - For storing temporal data

Choosing the Right Type

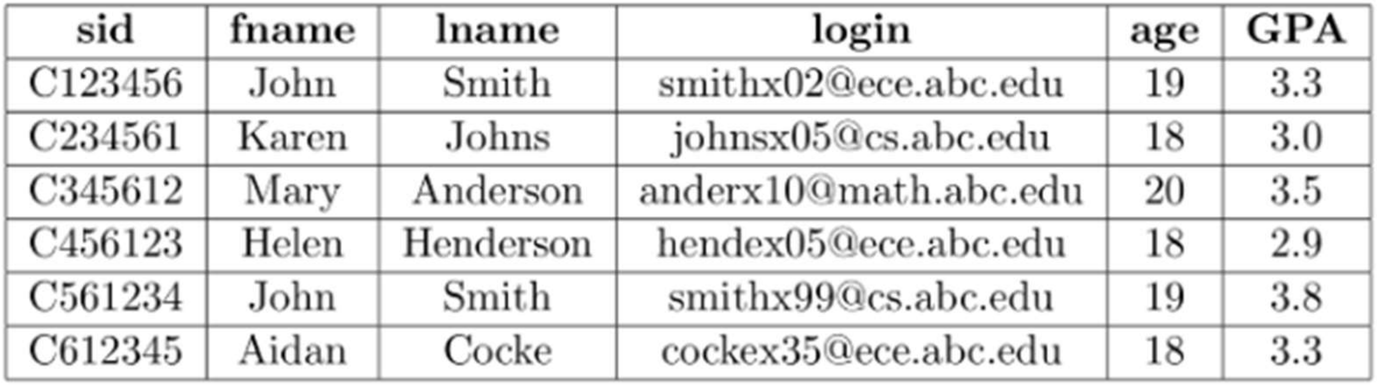
- Need to think carefully about what type makes most sense for your data values. Example: Storing ZIP codes
  - US postal codes for mail routing
  - 5 digits, e.g. 91125 for Caltech
- Does **INTEGER** make sense?
- **Problem 1**: Some ZIP codes have leading zeroes!
  - Many east-coast ZIP codes start with 0.
  - Numeric types won’t include leading zeros.
- **Problem 2**: US mail also uses ZIP+4 expanded ZIP codes
  - e.g. 91125-8000
- **Problem 3**: Many foreign countries use non-numeric values

Choosing the Right Type

- Better choice for ZIP codes?
  - A CHAR or VARCHAR column makes much more sense
- For example:
  - CHAR(5) or CHAR(9) for US-only postal codes
  - VARCHAR(20) for US + international postal codes
- Another example: monetary amounts
  - Floating-point representations cannot exactly represent all values
    - e.g. 0.1 is an infinitely-repeating binary decimal value
  - Use NUMERIC to represent monetary values

Activity

Create table student for the following schema:
Student (sid, fname, lname, login, age, GPA)



sid	fname	lname	login	age	GPA
C123456	John	Smith	smithx02@ece.abc.edu	19	3.3
C234561	Karen	Johns	johnsx05@cs.abc.edu	18	3.0
C345612	Mary	Anderson	anderx10@math.abc.edu	20	3.5
C456123	Helen	Henderson	hendex05@ece.abc.edu	18	2.9
C561234	John	Smith	smithx99@cs.abc.edu	19	3.8
C612345	Aidan	Cocke	cockex35@ece.abc.edu	18	3.3

INTEGRITY CONSTRAINTS

Example

Beers (name, manf)

Bars (name, addr, license)

Drinkers (name, addr, phone)

Likes (drinker, beer)

Sells (bar, beer, price)

Frequents (drinker, bar)

Underline = **key** (tuples cannot have the same value in all key attributes)

Declaring Keys

- An attribute or list of attributes may be declared PRIMARY KEY
  - Either says that no two tuples of the relation may agree in

Beers (name, manf) →

```
CREATE TABLE Beers (
        name CHAR(20) PRIMARY KEY,
        manf CHAR(20)
);
```

Declaring Multi-attribute Keys

- A key declaration can appear as element in the list of elements of a CREATE TABLE statement
- This form is essential if the key consists of more than one attribute

Sells (bar, beer, price) →

```
CREATE TABLE Sells (
          bar CHAR(20),
          beer VARCHAR(20),
          price REAL,
          PRIMARY KEY(bar, beer)
);
```

Foreign Keys

- Values appearing in attributes of one relation must appear together in certain attributes of another relation

Example:
We might expect that a value in **Sells.beer** also appears as value in **Beers.name**:

Beers(name, manf)
Sells(bar, beer, price)

Expressing Foreign Keys

- Use keyword **REFERENCES**, either:
  - After an attribute (for one-attribute keys)
    - **REFERENCES** <relation> (<attributes>)
  - As an element of the schema:
    - **FOREIGN KEY** (<list of attributes>)
    - **REFERENCES** <relation> (<attributes>)
- Referenced attributes must be declared **PRIMARY KEY**

Example: With Attribute

```
CREATE TABLE Beers (
        name CHAR(20) PRIMARY KEY,
        manf CHAR(20)
);

CREATE TABLE Sells (
        bar CHAR(20),
        beer VARCHAR(20) REFERENCES Beers(name),
        price REAL,
        PRIMARY KEY(bar, beer)
);
```

Example: As Schema Element

```
CREATE TABLE Beers (
        name CHAR(20) PRIMARY KEY,
        manf CHAR(20)
);
CREATE TABLE Sells (
        bar CHAR(20),
        beer VARCHAR(20),
        price REAL,
        PRIMARY KEY(bar, beer),
        FOREIGN KEY(beer) REFERENCES Beers(name)
);
```

Enforcing Foreign-Key Constraints

- If there is a foreign-key constraint from relation R to relation S, two violations are possible:
  - An insert or update to **R** introduces values not found in **S**
  - A deletion or update to **S** causes some tuples of **R** to “dangle”

Example: suppose for the two tables, **Sells** → **Beers**

- An insert or update to Sells that introduces a non-existent beer must be rejected
- A deletion or update to Beers that removes a beer value found in some tuples of Sells can be handled in three ways (next slide).

Actions Taken

- **DEFAULT**: Reject the modification
  - Deleted beer in **Beer**: set default value in **Sells** tuples
  - Updated beer in **Beer**: set default value in **Sells** tuples
- **CASCADE**: Make the same changes in **Sells**
  - Deleted beer in **Beer**: delete **Sells** tuple
  - Updated beer in **Beer**: change value in **Sells**
- **SET NULL**: Change the beer to NULL
  - Deleted beer in **Beer**: set NULL values in **Sells** tuples
  - Updated beer in **Beer**: set NULL values in **Sells** tuples

Example Sells → Beers

- **Delete the ‘Bud’ tuple from Beers**
  - **DEFAULT:** Do not change any tuple from Sells that have beer = ‘Bud’
  - **CASCADE:** Delete all tuples from Sells that have beer = ’Bud’
  - **SET NULL:** Change all tuples of Sells that have beer = ’Bud’ to have beer = NULL

- **Update the ‘Bud’ tuple to ’Budweiser’**
  - **DEFAULT**: do not change any tuple from Sells that have beer = ‘Bud’
  - **CASCADE**: change all Sells tuples with beer = ’Bud’ to beer = ’Budweiser’
  - **SET NULL**: Same change as for deletions

Choosing a Policy

- When we declare a foreign key, we may choose policies **SET NULL** or **CASCADE** independently for deletions and updates

- Follow the foreign-key declaration by:
  ON [UPDATE, DELETE][SET NULL, CASCADE]

- Two such clauses may be used, otherwise, the default (reject) is used.

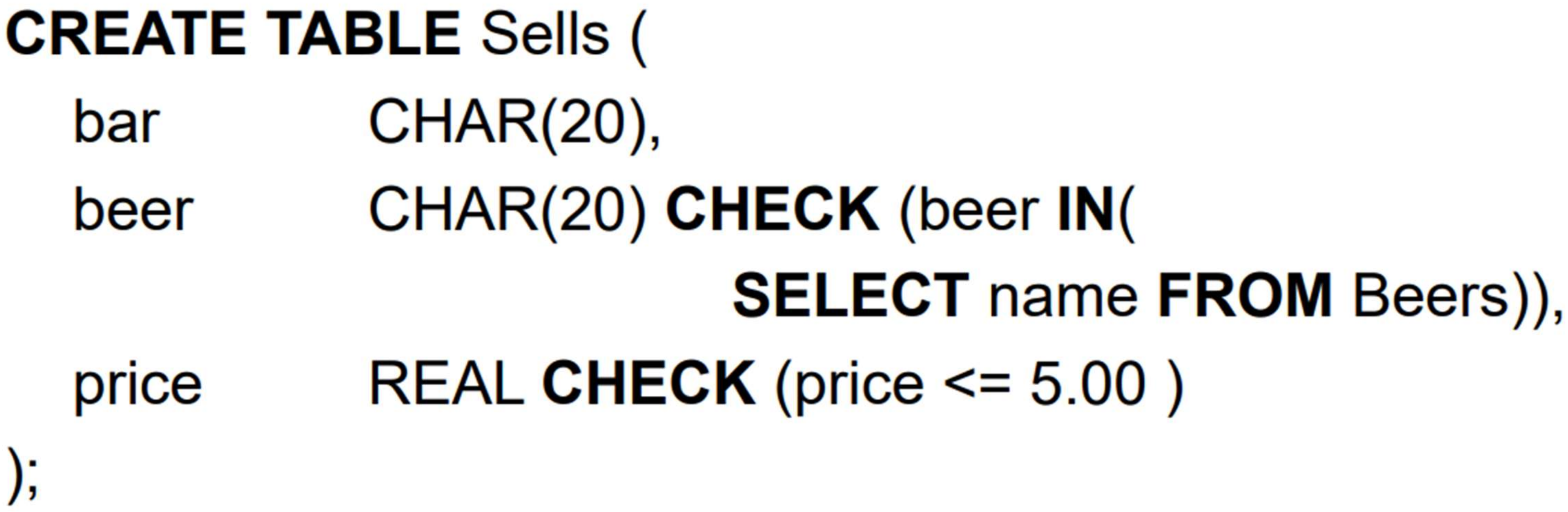
Example: Setting a Policy

```
CREATE TABLE Sells (
    bar          CHAR(20),
    beer         CHAR(20),
    price        REAL,
    FOREIGN KEY(beer) REFERENCES Beers(name)
                           ON DELETE SET NULL
                           ON UPDATE CASCADE
);
```

Attribute-Based Checks

- Constraints on the value of a particular attribute
- Add **CHECK**(<condition>) to the declaration for the attribute
- The condition may use the name of the attribute, but any other relation or attribute name must be in a subquery

Example: Attribute-Based Checks

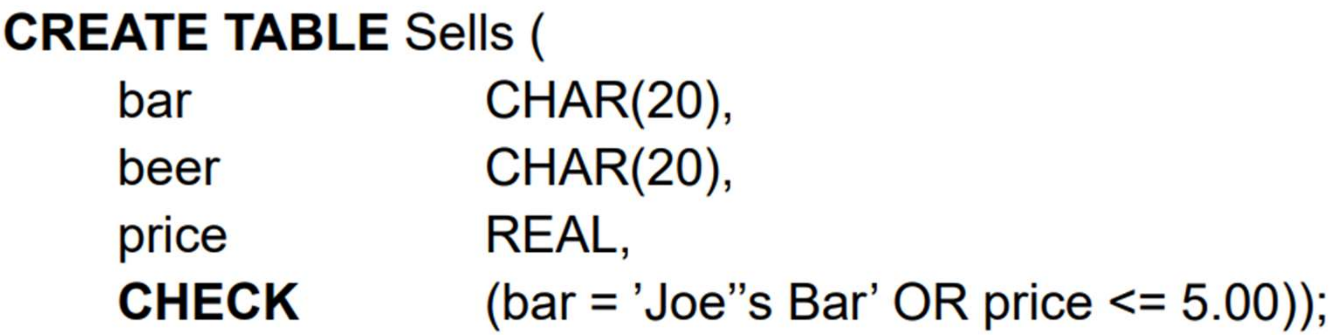


```
CREATE TABLE Sells (
    bar         CHAR(20),
    beer        CHAR(20) CHECK (beer IN(
                             SELECT name FROM Beers)),
    price       REAL CHECK (price <= 5.00 )
);
```

Tuple-Based Checks

- **CHECK** (<condition>) may be added as a relation schema element
  - The condition may refer to any attribute of the relation, but other attributes or relations require a subquery
  - Checked on insert or update only

- Example: Only Joe’s Bar can sell beer for more than $5



```
CREATE TABLE Sells (
    bar         CHAR(20),
    beer        CHAR(20),
    price       REAL,
    CHECK       (bar = ’Joe’’s Bar’ OR price <= 5.00));
```

DATA MANIPULATION LANGUAGE (DML)

Data Manipulation Language (DML)

- Syntax elements used for inserting, deleting and updating data in a database

- Modification statements include:
  - **INSERT** - for inserting data in a database
  - **DELETE** - for deleting data in a database
  - **UPDATE** - for updating data in a database

- All modification statements operate on a set of tuples (no duplicates)

Data Manipulation Language (DML)

- Example:
  - Employee (RegNo, FirstName, Surname, Dept, Office, Salary, City)
  - Department (DeptName, Address, City)
  - Product (Code, Name, Description, ProdArea)
  - LondonProduct (Code, Name, Description)

Insertions

Syntax varies:

- Using only values:

```
INSERT INTO Department
VALUES (‘Production’, ‘Rue du Louvre 23’, ’Toulouse’)
```

- Using both column names and values:

```
INSERT INTO Department(DeptName, City)
VALUES (‘Production’, ’Toulouse’)
```

- Using a subquery:

```
INSERT INTO LondonProducts
(SELECT Code, Name, Description
FROM Product
WHERE ProdArea = ‘London’)
```

Insertions

- The ordering of attributes (if present) and of values is meaningful -- first value for the first attribute, etc.
- If *AttributeList* is omitted, all the relation attributes are considered, in the order they appear in the table definition
- If *AttributeList* does not contain all the relation attributes, left-out attributes are assigned default values (if defined) or the NULL value

Activity

Movie (mID, title, year, director)

Reviewer (rID, name)

Rating (rID, mID, stars)

Activity

Movie

MID	TITLE	RELEASE_YEAR	DIRECTOR
101	Gone with the Wind	1939	Victor Fleming
102	Star Wars	1977	George Lucas
103	The Sound of Music	1965	Robert Wise
104	E.T.	1982	Steven Spielberg
105	Titanic	1997	James Cameron
106	Snow White	1937	-
107	Avatar	2009	James Cameron
108	Raiders of the Lost Ark	1981	Steven Spielberg

Activity

Reviewer

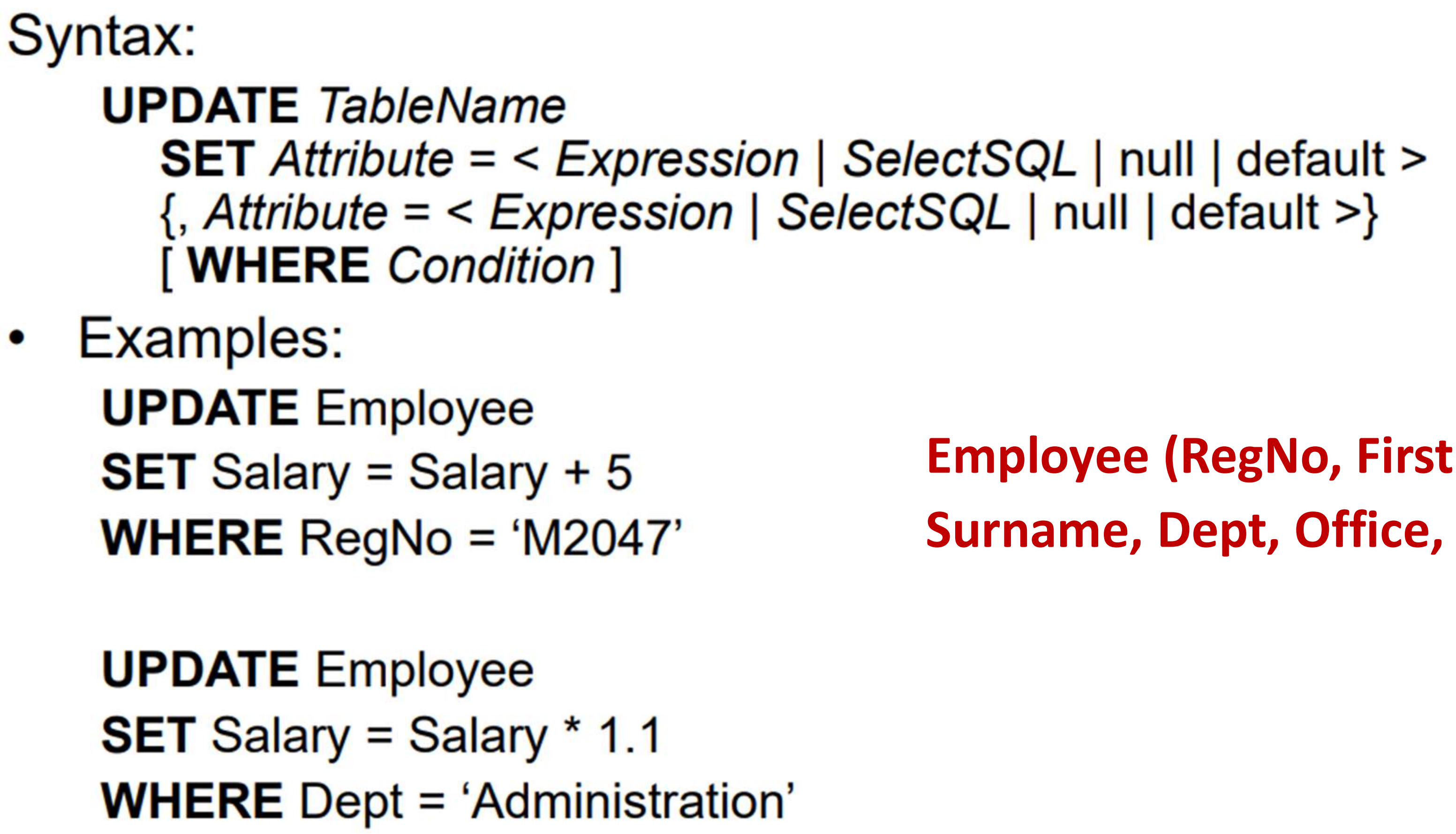
RID	NAME
201	Sarah Martinez
202	Daniel Lewis
203	Brittany Harris
204	Mike Anderson
205	Chris Jackson
206	Elizabeth Thomas
207	James Cameron
208	Ashley White

Activity

Rating

RID	MID	STARS
201	101	2
202	106	4
203	103	2
203	108	4
204	101	3
205	103	3
205	104	2
205	108	4
206	107	3
206	106	5
207	107	5
208	104	3

Update



Syntax:

```
UPDATE TableName
   SET Attribute = < Expression | SelectSQL | null | default >
   {, Attribute = < Expression | SelectSQL | null | default >}
   [ WHERE Condition ]
```

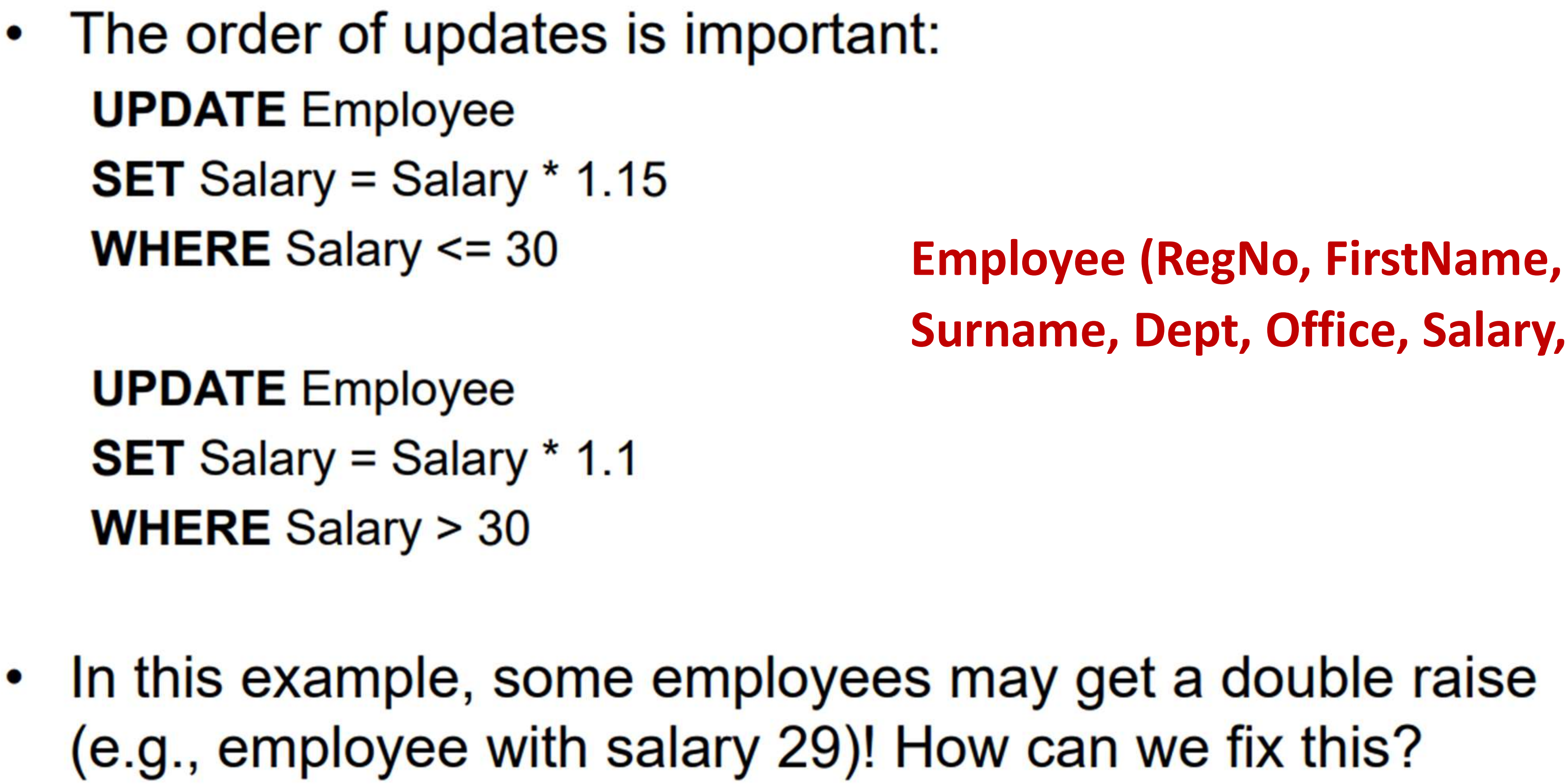
- Examples:

```
UPDATE Employee
SET Salary = Salary + 5
WHERE RegNo = 'M2047'

UPDATE Employee
SET Salary = Salary * 1.1
WHERE Dept = 'Administration'
```

Employee (RegNo, FirstName, Surname, Dept, Office, Salary, City)

Update



- The order of updates is important:

```
UPDATE Employee
SET Salary = Salary * 1.15
WHERE Salary <= 30

UPDATE Employee
SET Salary = Salary * 1.1
WHERE Salary > 30
```

- In this example, some employees may get a double raise (e.g., employee with salary 29)! How can we fix this?

Employee (RegNo, FirstName, Surname, Dept, Office, Salary, City)

Update

1.Update the director name of ‘Snow White’ to Robert Wise

2. Add 5 more years to the release year of all the movies released after 1970

MID	TITLE	RELEASE_YEAR	DIRECTOR
101	Gone with the Wind	1939	Victor Fleming
102	Star Wars	1977	George Lucas
103	The Sound of Music	1965	Robert Wise
104	E.T.	1982	Steven Spielberg
105	Titanic	1997	James Cameron
106	Snow White	1937	-
107	Avatar	2009	James Cameron
108	Raiders of the Lost Ark	1981	Steven Spielberg

Deletion

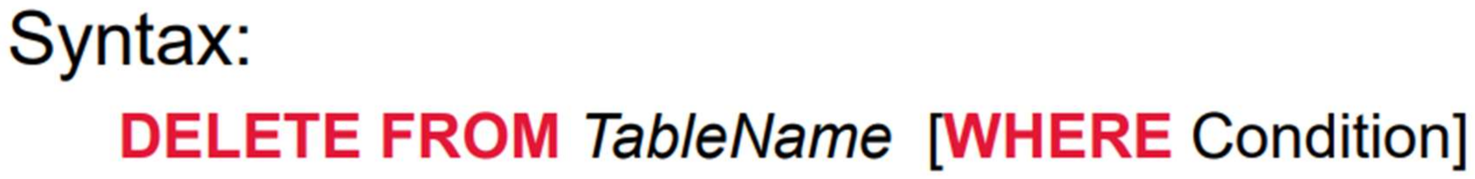
- The DELETE statement removes from a table all tuples that satisfy a condition
- If the WHERE clause is omitted, DELETE removes all tuples from the table (keeps the table schema):

```
DELETE FROM Department
```

- The removal may produce deletions from other tables – (see referential integrity constraint with cascade policy)
- To remove table Department completely (content and schema) :

```
DROP TABLE Department CASCADE
```

Deletion



Syntax:

```
DELETE FROM TableName [WHERE Condition]
```

1. Delete all movies with the title ‘Titanic’
2. Delete all movies with an MIDs greater than 106

MID	TITLE	RELEASE_YEAR	DIRECTOR
101	Gone with the Wind	1939	Victor Fleming
102	Star Wars	1977	George Lucas
103	The Sound of Music	1965	Robert Wise
104	E.T.	1982	Steven Spielberg
105	Titanic	1997	James Cameron
106	Snow White	1937	-
107	Avatar	2009	James Cameron
108	Raiders of the Lost Ark	1981	Steven Spielberg

SQL Queries

SQL Syntax

SELECT <desired attributes>
FROM <one or more tables>
WHERE <predicate holds for selected tuple>
GROUP BY <key columns, aggregations>
HAVING <predicate holds for selected group>
ORDER BY <columns to sort>

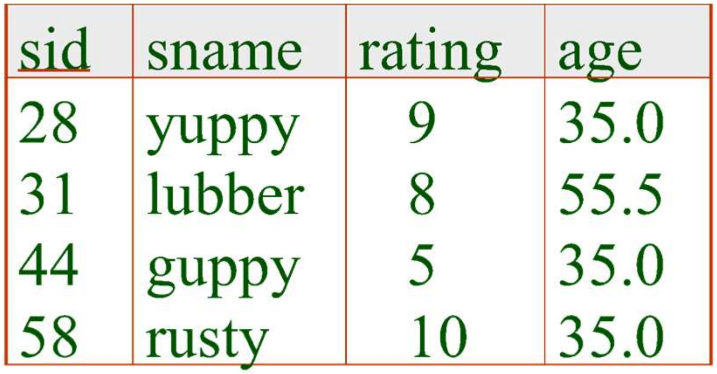
The SFW Query

SELECT <desired attributes>
FROM <one or more tables>
WHERE <predicate holds for selected tuple>

Projection (π)

Sailor (sid, sname, rating, age)

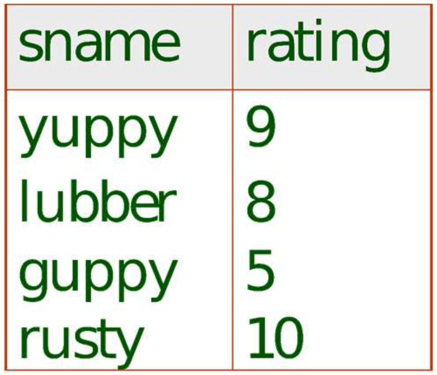
- **List the names and ratings of all the sailors**



sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

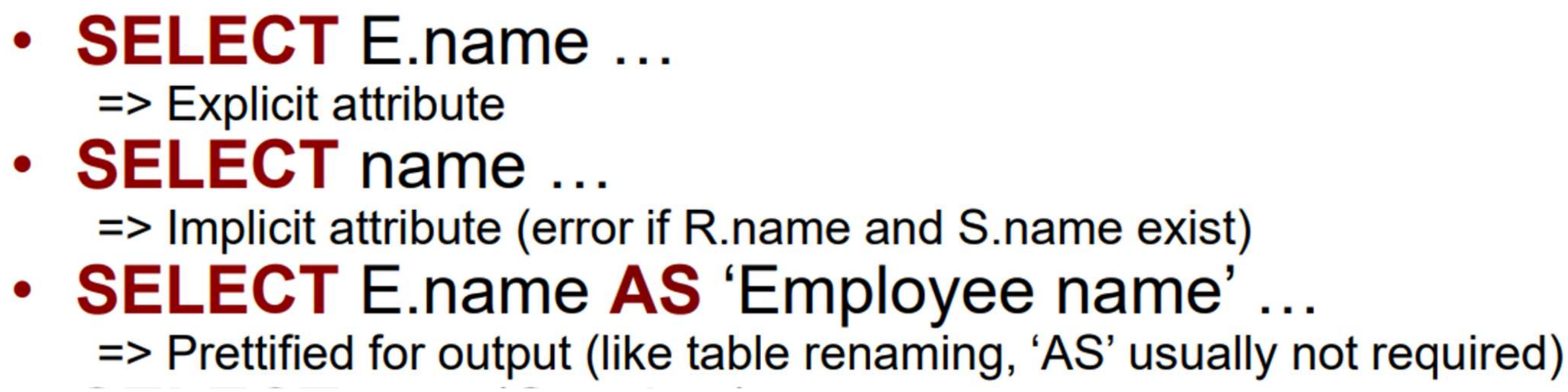
Sailor

$\pi_{sname,rating}$**(Sailor)**



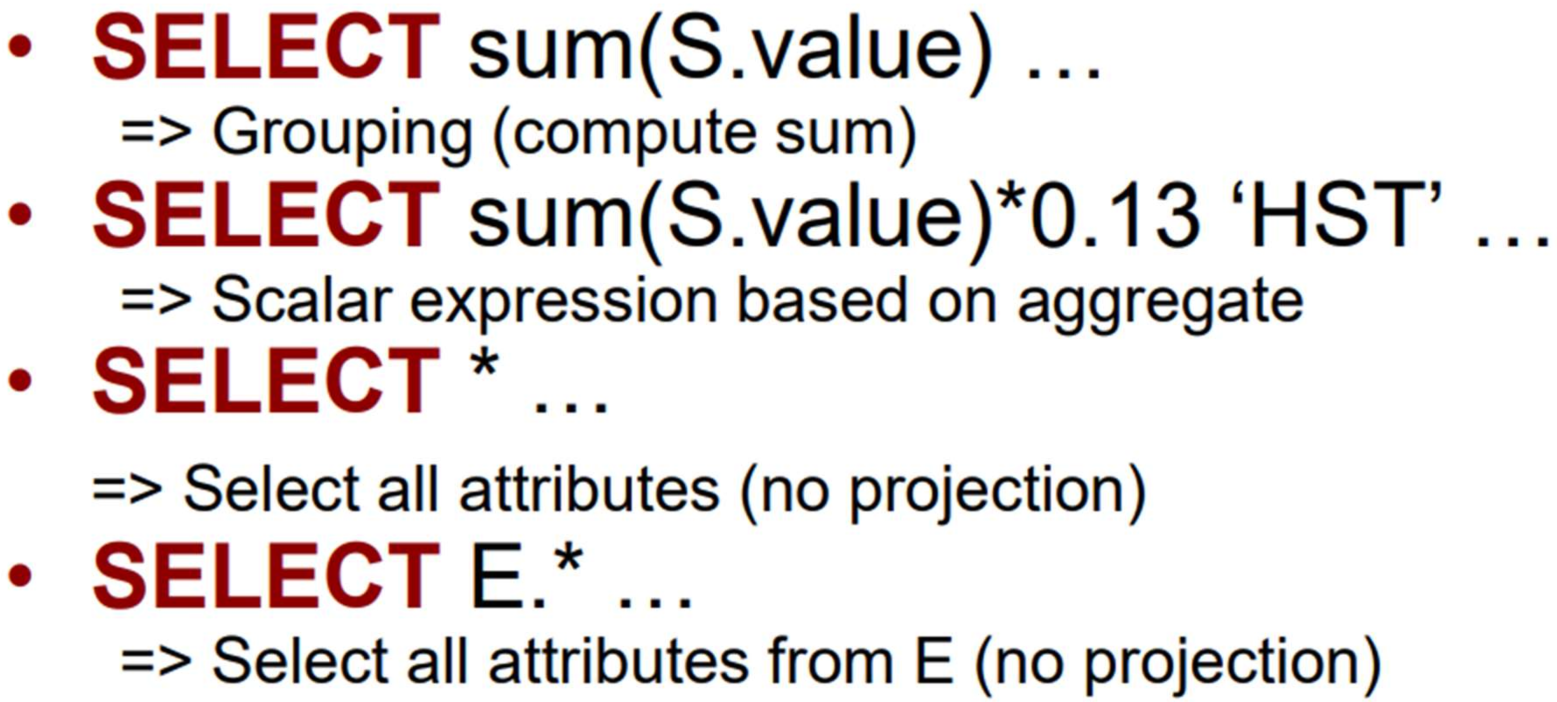
sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

The Select Clause



- **SELECT** E.name …
  => Explicit attribute
- **SELECT** name …
  => Implicit attribute (error if R.name and S.name exist)
- **SELECT** E.name **AS** ‘Employee name’ …
  => Prettified for output (like table renaming, ‘AS’ usually not required)

The Select Clause



- **SELECT** sum(S.value) …
  => Grouping (compute sum)
- **SELECT** sum(S.value)*0.13 ‘HST’ …
  => Scalar expression based on aggregate
- **SELECT** * …

  => Select all attributes (no projection)
- **SELECT** E.* …
  => Select all attributes from E (no projection)

The From Clause

Identifies the tables (relations) to query – Comma-separated list

- … **FROM** Employees

  => Explicit relation
- … **FROM** Employees **AS** E

  => Table alias (most systems don’t require “AS” keyword)
- … **FROM** Employees, Sales

  => Cartesian product

The From Clause: Join

- … **FROM** Employees E **JOIN** Sales S

  => Cartesian product (*no join condition given!*)
- … **FROM** Employees E **JOIN** Sales S **ON** E.EID=S.EID

  => Equi-join
- … **FROM** Employees **NATURAL JOIN** Sales

  => Natural join (bug-prone, use equijoin instead)
- … **FROM** Employees E
  LEFT JOIN Sales S **ON** E.EID=S.EID

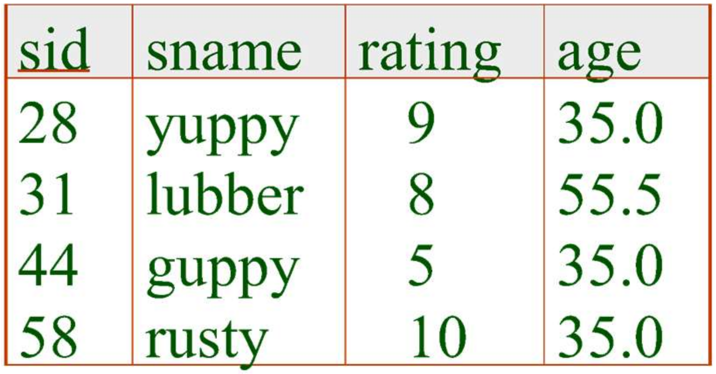
  => Left join
- … **FROM** Employees E1
  JOIN Employees E2 **ON** E1.EID < E2.EID

  => Theta self-join (*what does it return?*)

Selection (σ)

Sailor (sid, sname, rating, age)

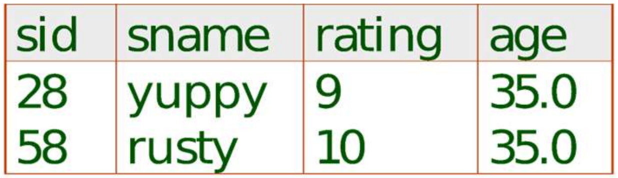
- **List the tuples where sailor rating is more than 8**



sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Sailor

$\sigma_{rating>8}$**(Sailor)**



sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

The Where Clause

- Conditions which all returned tuples must meet
  - Arbitrary boolean expression
  - Combine multiple expressions with **AND/OR/NOT**

- Often used instead of **JOIN**
  - **FROM** tables (Cartesian product, e.g. A, B)
  - Specify join condition in **WHERE** clause (e.g. A.ID=B.ID)

The Where Clause

- … **WHERE** S.date > ‘01-Jan-2010’

  => Simple tuple-literal condition
- … **WHERE** E.EID = S.EID

  => Simple tuple-tuple condition (equi-join)
- … **WHERE** E.EID = S.EID **AND** S.PID = P.PID

  => Conjunctive tuple-tuple condition (three-way equijoin)
- … **WHERE** S.value < 10 **OR** S.value > 10000

  => Disjunctive tuple-literal condition

Pattern matching

- … **WHERE** phone **LIKE** ‘%268-_ _ _ _’
  - phone numbers with exchange 268
  - WARNING: spaces are wrong, only shown for clarity
- … **WHERE** last_name **LIKE** ‘Jo%’
  - Jobs, Jones, Johnson, Jorgensen, etc.
- … **WHERE** Dictionary.entry **NOT LIKE** ‘%est’
  - Ignore ‘biggest’, ‘tallest’, ‘fastest’, ‘rest’, …
- … **WHERE** sales **LIKE** '%30!%%' **ESCAPE** ‘!’
  - Sales of 30%

Use of Limit clause

Student (sid, sname, age, city)

- Select the first two tuples from the student table
  - Select * from student limit 2;

- Find the sid of the youngest student
  - Select sid from student order by age limit 1;

Find names of sailors who’ve reserved boat #103

Sailors

Sid	Sname	Rating	Age
28	Yuppy	9	35
31	Lubber	8	55
44	Guppy	5	35
58	Rusty	10	35

Boats

Bid	Bname	Color
101	Interlake	Blue
102	Interlake	Red
103	Clipper	Green
104	Marine	Red

Reserves

Sid	Bid	Day
22	101	10/10/23
58	103	11/12/23

Practice 1

Employee(RegNo, FirstName, Surname, Dept_id, Office, Salary, City)
Department(DeptId, DeptName, Address, City)

1. Find the salaries of employees with surname Brown
2. Find all the information relating to employees with surname Brown
3. Find the monthly salary of employees sur-named White
4. Find the names of employees and their cities of work
5. Find the first names and surnames of employees who work in office number 20 of the Administration department
6. Find the first names and surnames of employees who work in either the Administration or the Production department
7. Find the first names of employees named Brown who work in the same city that they live in
8. Find employees with surnames that have ‘r’ as the second letter and end in ‘n’

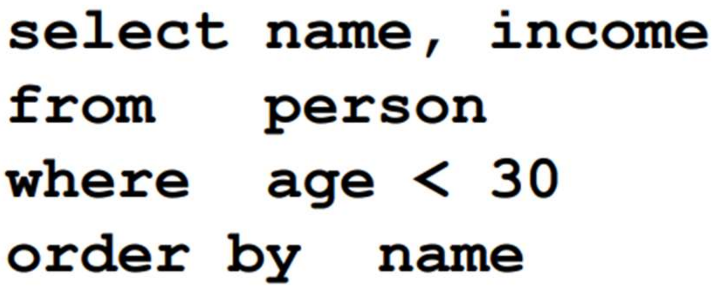
Practice 2

employee(empid, empname, departmentid, contactno, email, empheadid)
empdept(deptid, deptname, dept_offday, deptheadid)
empsalary(empid, salary, ispermanent)
project(projectid, duration)
country(cid, cname)
clienttable(clientid, clientname, cid)
empproject(empid, projectid, clientid, startyear, endyear)

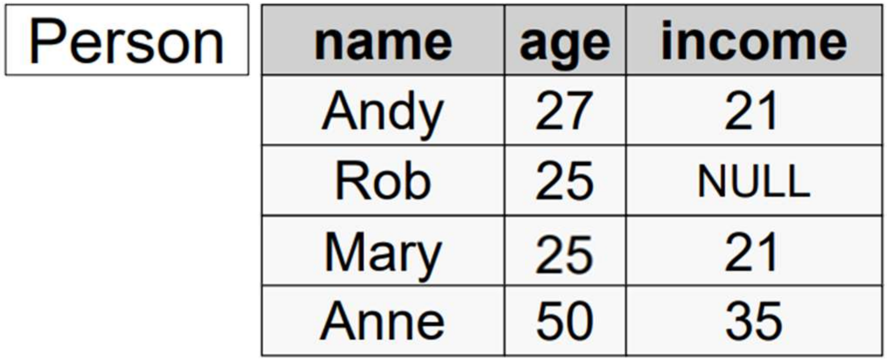
1. Select the department names of the employees whose salary is more than 140000.
2. Select the name of the employee who is working under Adam
3. Select the name of the employee who is department head of the HR department.
4. Select the names of the employees who are permanent
5. Select the name and email of the Dept Head who is not Permanent
6. Select the employee whose department off is ‘Monday’
7. select the details of the Indian clients.
8. select the details of all employee working in Development department
9. List the employee names who work in projects that have client from England
10. select the name of the employee whose name’s 2nd character is ‘h’, work as a department head and make salary more than 160000;

Ordering the Result: order by

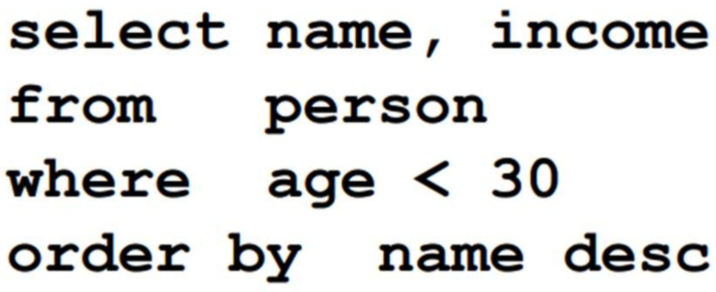
- “Return name and income of persons under thirty, in alphabetic order of the names”



```
select name, income
from   person
where  age < 30
order by  name
```



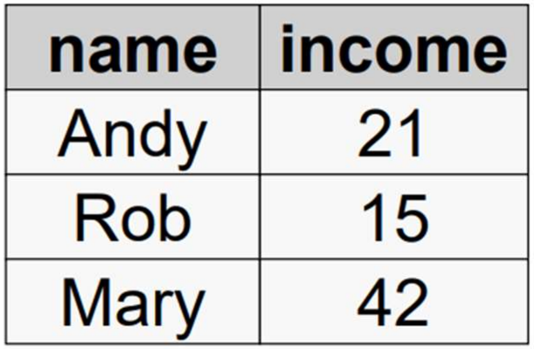
Person	name	age	income
	Andy	27	21
	Rob	25	NULL
	Mary	25	21
	Anne	50	35



```
select name, income
from   person
where  age < 30
order by  name desc
```

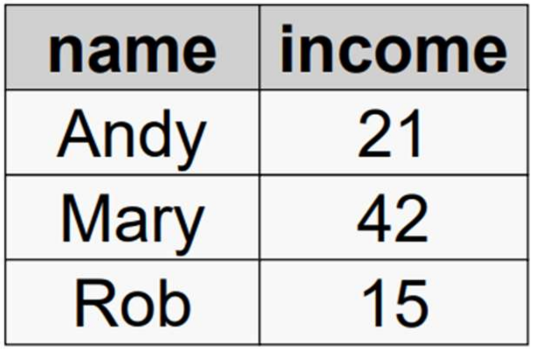
Ordering the Result: order by

```
select name, income
from   person
where  age < 30
```



name	income
Andy	21
Rob	15
Mary	42

```
select name, income
from   person
where  age < 30
order by name
```



name	income
Andy	21
Mary	42
Rob	15

Aggregate Functions

Aggregate Functions

- Aggregate functions:
  - **SUM** sums the values in the collection
  - **AVG** computes average of values in the collection
  - **COUNT** counts number of elements in the collection
  - **MIN** returns minimum value in the collection
  - **MAX** returns maximum value in the collection

- **SUM** and **AVG** require numeric inputs (obvious)
- Basic Syntax (simplified):

  Function ([distinct] ExpressionOnAttributes)

Practice : Employee

Role	Name	Building	Years_employed
Engineer	Becky A.	1e	4
Engineer	Dan B.	1e	2
Engineer	Sharon F.	1e	6
Engineer	Dan M.	1e	4
Engineer	Malcom S.	1e	1
Artist	Tylar S.	2w	2
Artist	Sherman D.	2w	8
Artist	Jakob J.	2w	6
Artist	Lillia A.	2w	7
Artist	Brandon J.	2w	7
Manager	Scott K.	1e	9
Manager	Shirlee M.	1e	3
Manager	Daria O.	2w	6

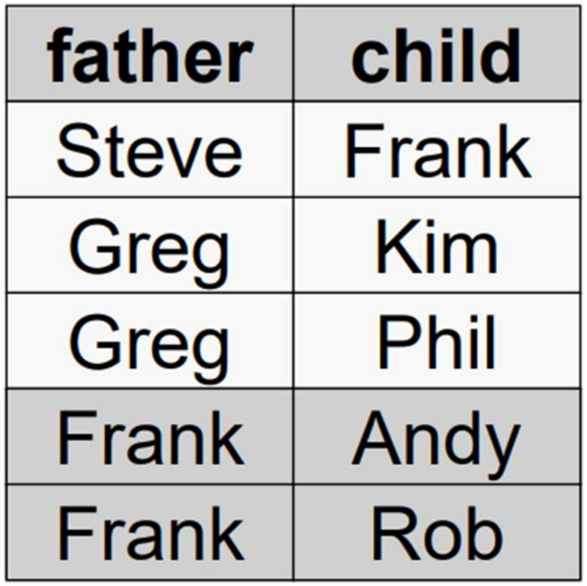
- **How many Beckys are engineers?**
- **How many employees work in building 1e?**
- **Average experience of engineers?**
- **Who is the manager with least experience?**

Aggregate Functions: count

- Syntax:
- counts the number of tuples:
  - count (*)
- counts the values of an attribute (considering duplicates):
  - count (Attribute)
- counts the distinct values of an attribute:
  - count (distinct Attribute)

Aggregate Functions: count

- Example: How many children Frank has?

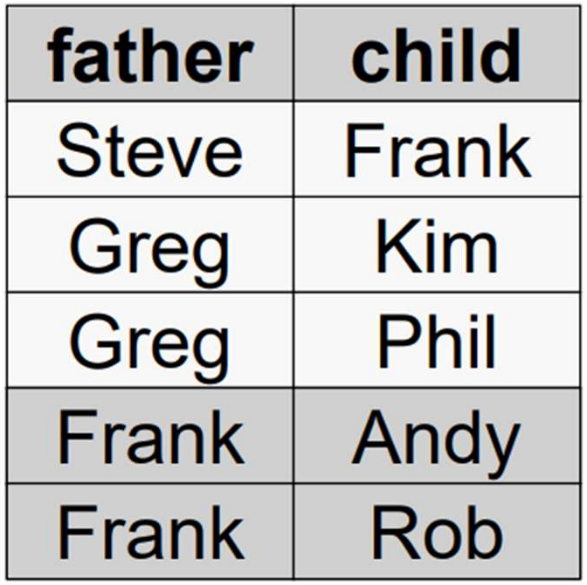


father	child
Steve	Frank
Greg	Kim
Greg	Phil
Frank	Andy
Frank	Rob

Aggregate Functions: count

- Example: How many children Frank has?

```
select * from fatherChild
      where father = 'Frank'
```

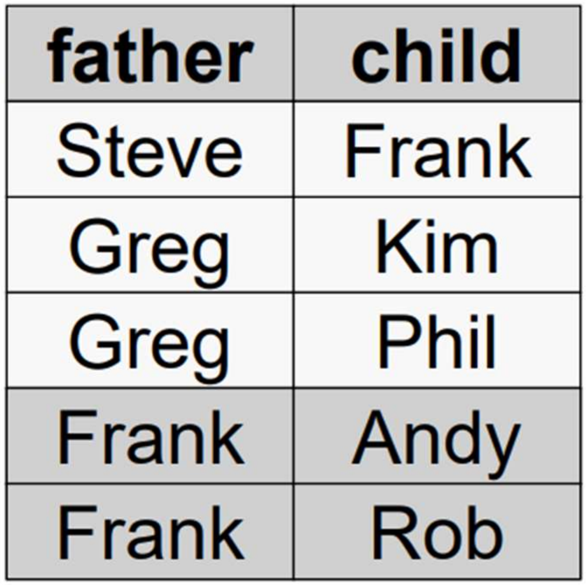


father	child
Steve	Frank
Greg	Kim
Greg	Phil
Frank	Andy
Frank	Rob

Aggregate Functions: count

- Example: How many children Frank has?

```
select count(*) as NumFranksChildren
   from fatherChild
        where father = 'Frank’
```

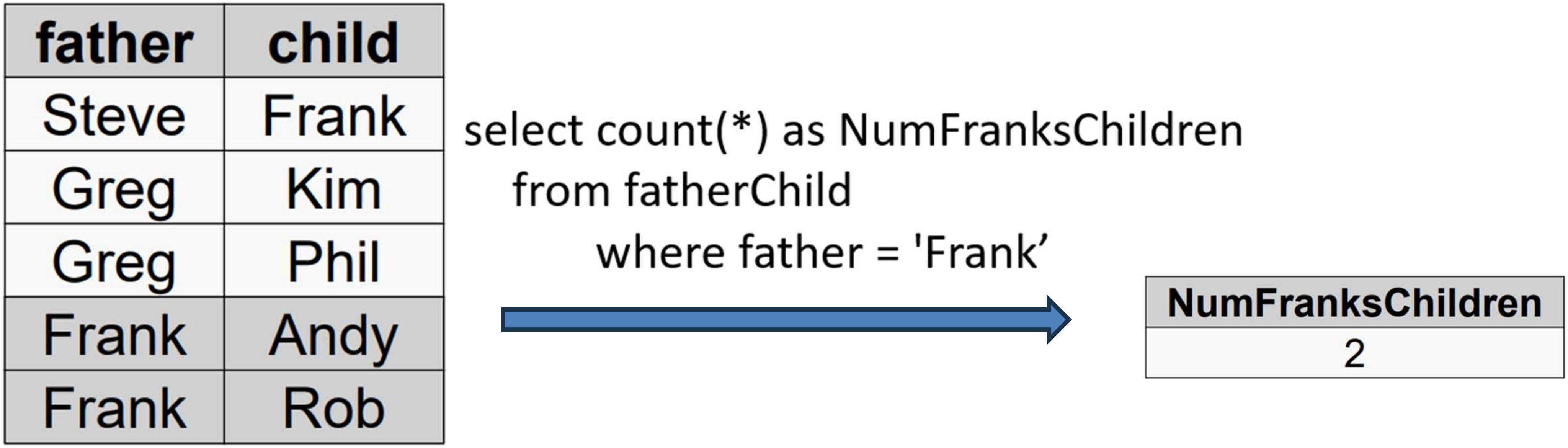


father	child
Steve	Frank
Greg	Kim
Greg	Phil
Frank	Andy
Frank	Rob

- Semantics: The aggregate operator (count), which counts the tuples, is applied to the result of the query:

```
select * from fatherChild
     where father = 'Frank'
```

Aggregate Functions: count



father	child
Steve	Frank
Greg	Kim
Greg	Phil
Frank	Andy
Frank	Rob

```
select count(*) as NumFranksChildren
   from fatherChild
         where father = 'Frank’
```

NumFranksChildren
2

Aggregate Functions: count and Null Values

```
select  count(*)
from    person
```

Result = number of tuples
= 4

```
select  count(income)
from    person
```

Result = number of values different from NULL
= 3

```
select  count(distinct income)
from    person
```

Result = number of distinct values (excluding NULL)
= 2

Person

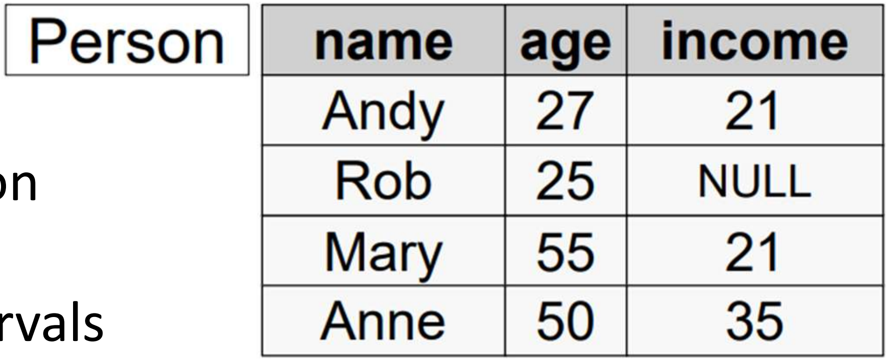
name	age	income
Andy	27	21
Rob	25	NULL
Mary	55	21
Anne	50	35

Other Aggregate Operators

sum, avg, max, min

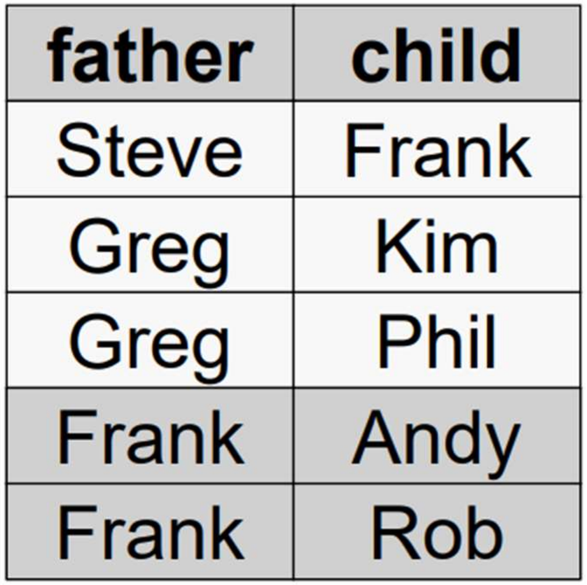
- Argument can be an attribute or an expression (but not “ *”)
- **sum** and **avg**: of numeric types and time intervals
- **max** and **min**: on types for which an ordering is defined: numbers, strings, time intervals, arrays

Example: **Average income of Frank’s children**



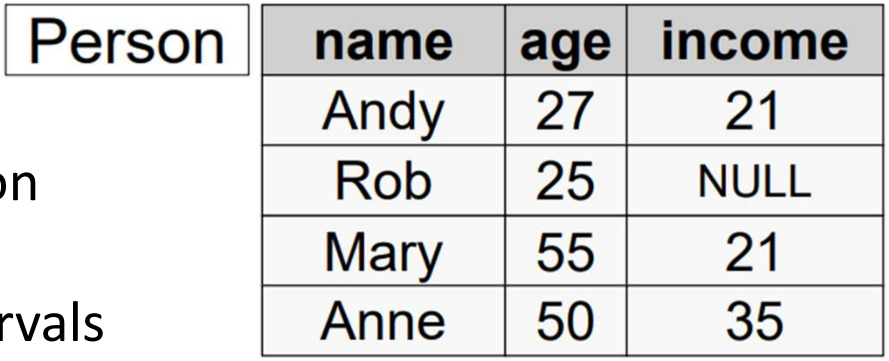
Person

name	age	income
Andy	27	21
Rob	25	NULL
Mary	55	21
Anne	50	35



father	child
Steve	Frank
Greg	Kim
Greg	Phil
Frank	Andy
Frank	Rob

Other Aggregate Operators



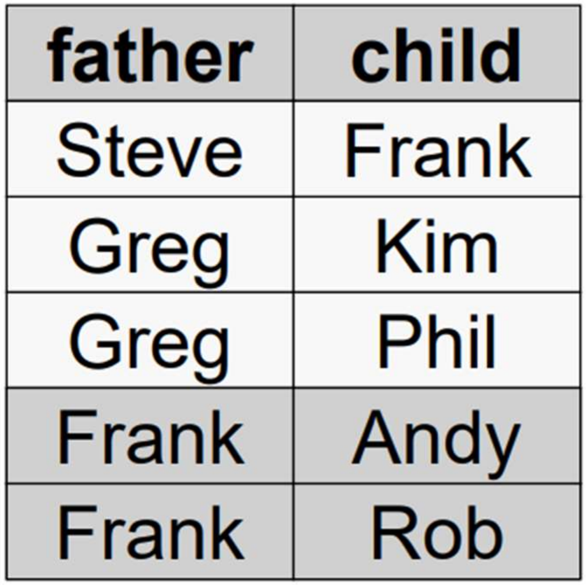
Person	name	age	income
	Andy	27	21
	Rob	25	NULL
	Mary	55	21
	Anne	50	35

sum, avg, max, min

- Argument can be an attribute or an expression (but not “ *”)
- **sum** and **avg**: of numeric types and time intervals
- **max** and **min**: on types for which an ordering is defined: numbers, strings, time intervals, arrays

Example: **Average income of Frank’s children**

```
select avg(p.income)
  from person p, fatherChild fc
     where on p.name = fc.child  and fc.father = 'Frank'
```



father	child
Steve	Frank
Greg	Kim
Greg	Phil
Frank	Andy
Frank	Rob

Aggregate Functions and Null Values

```
select avg(income) as meanIncome
from   person
```

Person	name	age	income
	Andy	27	30
	Rob	25	NULL
	Mary	55	36
	Anne	50	36

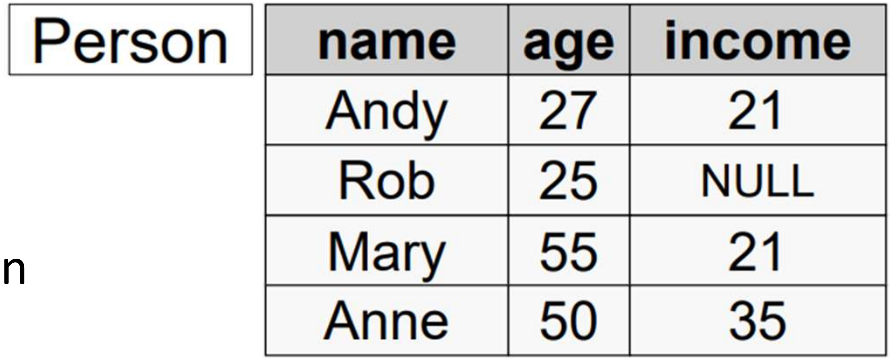
is
ignored

meanIncome
34

Aggregate Functions and the Target List

- An incorrect query (whose name should be returned?):

  select name, **max**(income) from person



Person

name	age	income
Andy	27	21
Rob	25	NULL
Mary	55	21
Anne	50	35

- The target list has to be homogeneous, for example:

  select **min**(age), **avg**(income) from person

- Aggregate functions compute a single value from a multiset of inputs
  - Doesn’t make sense to combine individual attributes and aggregate functions like this

Set Operations

Relational *Instance* **S1**

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

Relational *Instance* **S2**

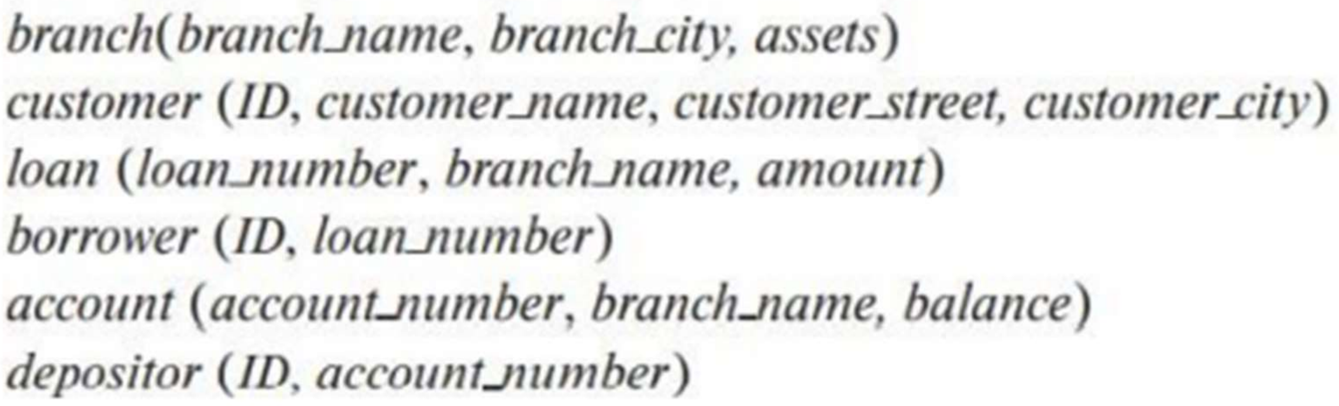
sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

(select * from *S1*)
union
(select * from *S2)*

(**select** *sname* **from** *S1*)
intersect
(select *sname* **from** *S2)*

(**select** *sname* **from** *S1*)
except
(select *sname* **from** *S2)*

Set Operations



branch(branch_name, branch_city, assets)
customer (ID, customer_name, customer_street, customer_city)
loan (loan_number, branch_name, amount)
borrower (ID, loan_number)
account (account_number, branch_name, balance)
depositor (ID, account_number)

Find all customers who have both a loan and an account.

Find all customers who have a loan, an account, or both:

Find all customers who have an account but no loan.

Set Operations

Find all customers who have a loan, an account, or both:

(select *customer_name* **from** *depositor*)
union
(select *customer_name* **from** *borrower)*

Find all customers who have both a loan and an account.

(**select** *customer_name* **from** *depositor*)
intersect
(select *customer_name* **from** *borrower)*

Find all customers who have an account but no loan.

(**select** *customer_name* **from** *depositor*)
except
(select *customer_name* **from** *borrower)*