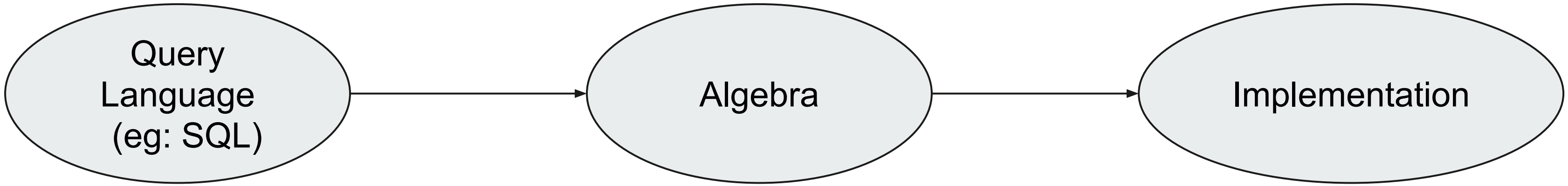
RELATIONAL ALGEBRA

What is Relational Algebra

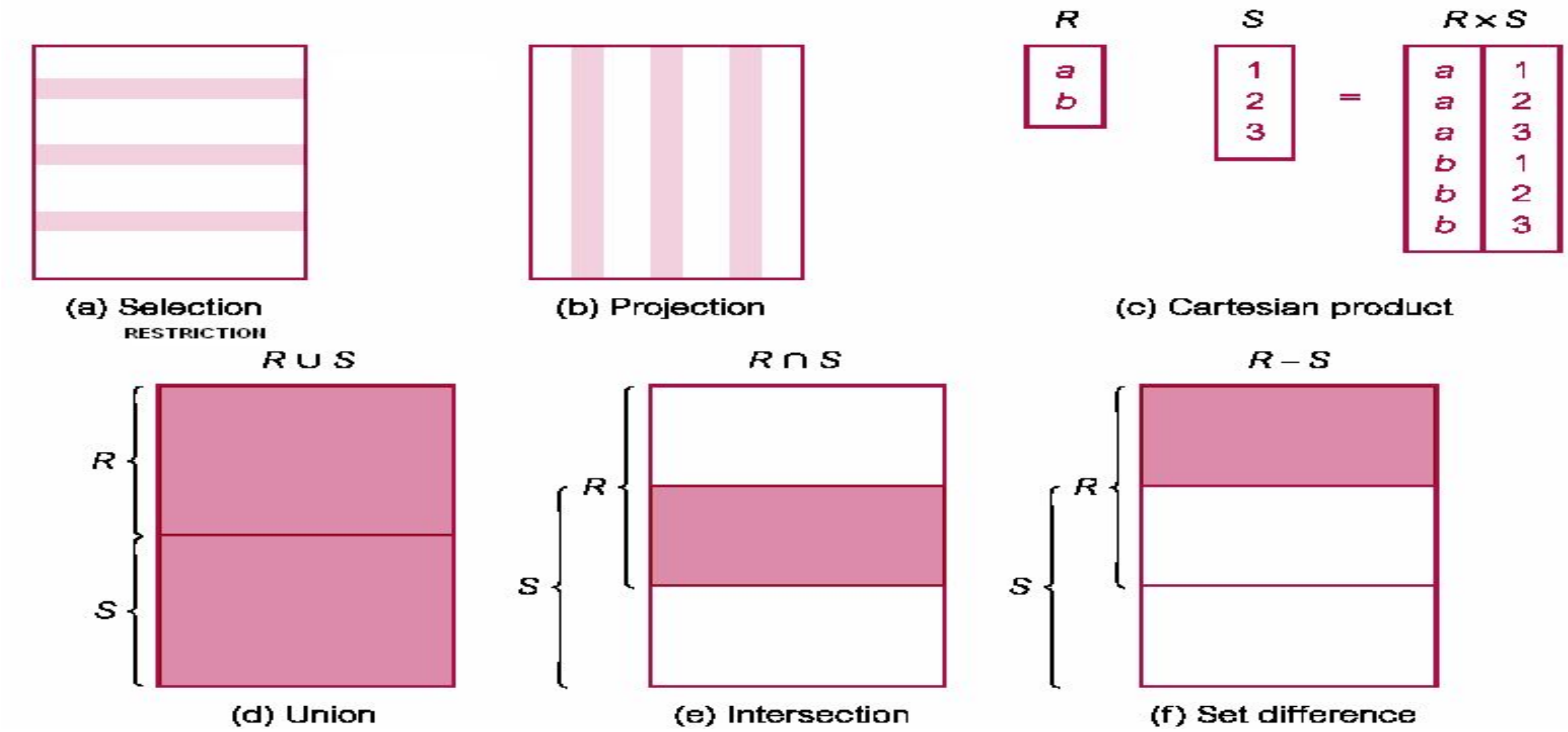
- It is a language in which we can ask questions (query) of a database.
- Formalism for creating new relations from existing ones.



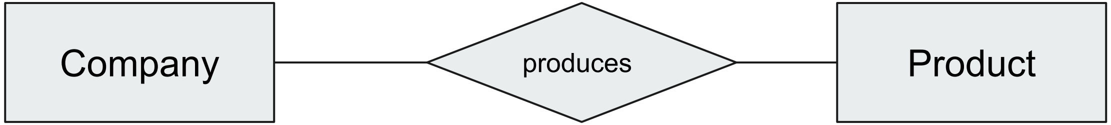
What is Relational Algebra

- Relational algebra is a procedural query language.
- It consists of the select, project, union, set difference, Cartesian product, and rename operations.
- Set intersection, division, natural join, and assignment combine the fundamental operations.
- SQL is based on relational algebra

Operations


R
a
b
S
1
2
3
=
R × S
a 1
a 2
a 3
b 1
b 2
b 3
(a) Selection
RESTRICTION
(b) Projection
(c) Cartesian product
R ∪ S
R
S
(d) Union
R ∩ S
R
S
(e) Intersection
R − S
R
S
(f) Set difference

Example



Company

<u>CName</u>	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Product

<u>PName</u>	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

SEMANTICS OF THE SAMPLE RELATIONS:

Company: Entity set; lists the relevant properties of company.

Product: Entity set; lists the relevant properties of product.

Produces: Relationship set: links company and product by describing the cname and pname.

SQL Queries

Creating Relations : Schema, Instance

- CREATE TABLE Student (sid CHAR(20), name VARCHAR(20), age INTEGER, gpa FLOAT);

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

Destroying Relations

- DROP TABLE Student;

NULL

- INSERT INTO Student (sid, gpa) VALUES (53689, 3.5);
- ALTER TABLE Student MODIFY name VARCHAR(20) NOT NULL;

DATA MANIPULATION LANGUAGE

- INSERT INTO Student (sid, name,login, age, gpa) VALUES (53688, 'Smith', 'ds@rutgers.edu', 18, 3.2);
- UPDATE Student SET age = 19 WHERE sid = 53688;
- DELETE FROM Student WHERE sid = 53688 ;

Integrity Constraints (ICs)

- ➔ IC's are specified when schema is defined.
- ➔ IC's are checked when relations are modified.

- domain integrity, entity integrity, referential integrity, foreign key ic.
  - **Domain Integrity** means the definition of a valid set of values for an attribute. You define - data type, - length or size, - is null value allowed, - is the value unique or not for an attribute.
  - **Entity integrity** constraint states that primary keys can't be null.

Integrity Constraints (ICs) Contd.

CREATE TABLE Enrolled (sid CHAR(20) cid CHAR(20), year INT, term INT grade CHAR(2), PRIMARY KEY (sid, cid, year, term))

A student can retake a course.

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

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

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

A student can only take one course.

Candidate Key vs Primary Key

Candidate Key – A Candidate Key can be any column or a combination of columns that can qualify as unique key in database. There can be multiple Candidate Keys in one table. Each Candidate Key can qualify as Primary Key.

Primary Key – A Primary Key is a column or a combination of columns that uniquely identify a record. Only one Candidate Key can be Primary Key.

Integrity Constraints (ICs) Contd.

- **Referential Integrity Constraint** constraint is specified between two tables and it is used to maintain the consistency among rows between the two tables. The rules are:
  - You can't delete a record from a primary table if matching records exist in a related table.
  - You can't change a primary key value in the primary table if that record has related records.
  - You can't enter a value in the foreign key field of the related table that doesn't exist in the primary key of the primary table.
  - **However, you can enter a Null value in the foreign key, specifying that the records are unrelated.**

Integrity Constraints (ICs) Contd.

- **Foreign Key Integrity Constraint :** cascade update related fields and cascade delete related rows. These constraints affect the referential integrity constraint.
  - Cascade Update Related Fields (on update cascade) : Any time you change the primary key of a row in the primary table, the foreign key values are updated in the matching rows in the related table. This constraint overrules rule 2 in the referential integrity constraints.

create table EnrolledIn (sid INT references Students, cid INT references Courses, grade CHAR (2), primary key(sid,cid))

Integrity Constraints (ICs) Contd.

- Cascade Delete Related Rows (on delete cascade): Any time you delete a row in the primary table, the matching rows are automatically deleted in the related table. This constraint overrules rule 1 in the referential integrity constraints.

SQL SubQueries

Subqueries with the SELECT Statement

Syntax :

SELECT column_name [, column_name] FROM   table1 [, table2] WHERE column_name OPERATOR   (SELECT column_name [, column_name]
FROM table1 [, table2]
[WHERE]) ;

Example :

“https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_in”

SELECT * from Orders where OrderID in (SELECT OrderID FROM OrderDetails WHERE Quantity > 70);

Subqueries with the INSERT Statement

Syntax :

INSERT INTO table_name [(column1 [, column2])] SELECT [*|column1 [, column2]
FROM table1 [, table2]   [WHERE VALUE OPERATOR];

Example :

“https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_in”

INSERT INTO CUSTOMERS_BKP   SELECT * FROM CUSTOMERS
WHERE ID IN (SELECT ID FROM CUSTOMERS) ;

Subqueries with the UPDATE Statement

Syntax :

UPDATE table SET column_name = new_value [WHERE OPERATOR [VALUE]

(SELECT COLUMN_NAME FROM TABLE_NAME) [WHERE)];

Example :

"https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_in"

UPDATE CUSTOMERS set date = '15-JAN-10'

WHERE ID IN (SELECT ID FROM CUSTOMERS) ;

Subqueries with the DELETE Statement

Syntax :

DELETE FROM TABLE_NAME [WHERE OPERATOR [VALUE]   (SELECT COLUMN_NAME   FROM TABLE_NAME)   [WHERE)];

Example :

“https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_in”

DELETE FROM CUSTOMERS WHERE ID IN (SELECT ID   FROM CUSTOMERS) ;

Rules for SubQuery

There are a few rules that subqueries must follow –

- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
- An ORDER BY command cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY command can be used to perform the same function as the ORDER BY in a subquery.
- Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.
- The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.
- A subquery cannot be immediately enclosed in a set function.
- The BETWEEN operator cannot be used with a subquery. However, the BETWEEN operator can be used within the subquery.

HWK- 4

Q1

SQL :

Select distinct(pub) from likes as l inner join serves as s on l.beer = s.beer where l.drinker='Joe';

Select distinct(pub) from serves where beer in (select beer from likes where drinker='Joe');

RA : Combine(drinker, beer, pub, cost) := LIKES ⋈ SERVES

FindJoe(drinker, beer, pub, cost) := σdrinker = 'Joe' (Combine)

Pubs(pub) := πpub (FindJoe)

Datalog : LIKES('Joe', Beer), SERVES(Pub, Beer, _).

Q2

SQL :

Select f.drinker from frequents as f inner join serves as s on f.pub = s.pub where s.cost < 3;

Select f.drinker from frequents where pubs in (select pubs from serves where cost < 3);

RA : Combine(drinker, pub, beer, cost) := FREQUENTS ⋈ SERVES

FindCheap(drinker, pub, beer, cost) := σcost < 3 (Combine)

Drinkers(drinker) := πdrinker (FindCheap)

Datalog : FREQUENTS(Drinker, Pub), SERVES(Pub, _, Cost), Cost < 3

Q3

SQL :

Select drinker from likes as l inner join (select l.beer from likes as l inner join serves as s on l.beer = s.beer where s.cost > 8 and l.drinker=joe) as y on l.beer=y.beer;

Select drinker from likes where beer in (select beer from likes as l inner join serves as s on l.beer = s.beer where s.cost > 8 and l.drinker='Joe');

Q3 Contd.

RA : Combine(drinker, beer, pub, cost) := LIKES ⋈ SERVES

FindJoe(drinker, beer, pub, cost) := σdrinker = ‘Joe’ AND cost > 8 (Combine)

Joe(beer) := πbeer (FindJoe)

//Others that like the same beer

MatchBeer(drinker, beer) := LIKES ⋈ Joe

Others(drinker) := πdrinker (MatchBeer)

Simple : π drinker (likes ⋈ π beer (σ drinker='Joe' and cost > 8 (likes ⋈ serves)))

Q4

SQL :

select likes.drinker from likes left join frequents on likes.drinker=frequents.drinker where frequents.drinker=null;

RA : LikeDrinker(drinker) := пdrinker (LIKES)

FrequentDrinker(drinker) := пdrinker (FREQUENTS)

Answer(drinkers) := LikeDrinker – FrequentDrinker

Complex : п likes.drinker σ frequents.drinker = null likes ⟕ likes.drinker = frequents.drinker frequents

Datalog : LIKES(Drinker, _), NOT FREQUENTS(Drinker, _).

Q5

SQL :

Select drinker from frequents as f inner join serves as s on f.pub = s.pub where beer='Stella Artois' or beer='Molsons';

Select drinkers from serves where pub in (select pub from serves where beer='Stella Artois' or beer='Molsons');

RA : Combine(drinker, pub, beer, cost) := FREQUENTS ⋈ SERVES

FindStella(drinker, pub, beer, cost) := σbeer_='Stella Artois' (Combine)

FindMolsons(drinker, pub, beer, cost) := σbeer_='Molsons' (Combine)

Pubs(drinker, pub, beer, cost) := FindStella ∪ FindMolsons

Drinkers(drinker) := πdrinker (Pubs

Q6

RA :

//Beer that Joe likes

FindJoe(drinker, beer) := σdrinker = ‘Joe’ (LIKES)

Joe(beer) := πbeer (FindJoe)

//The beers that pubs serve

PubBeer(pub, beer) := πpub, beer (SERVES)

//Use division between PubBeer and Joe

//Cross the pub column of PubBeer with Joe to get all possible tuples

AllPossible(pub, beer) := πpub(PubBeer) × Joe

Q6 Contd.

//Difference between AllPossible and PubBeer gives disqualified tuples. Project the pubs.

Disqualified(pub) := πpub(AllPossible – PubBeer)

//Qualified: Find the difference between the pub column of PubBeer and Disqualified

Answer(pub) := πpub(PubBeer) – Disqualifed

Datalog :

//someNotServe: Pubs that do not serve a beer that Joe likes

//In other words: Joe likes a beer, and the pub doesn't serve it.
someNotServe(Pub) :- NOT SERVES(Pub, Beer, _), LIKES('Joe', Beer).
answer(Pub) :- SERVES(Pub, _, _), NOT someNotServe(Pub).

Q7

RA :

//Drinkers like the beer. Pub serves the beer. Drinker frequents the pub.

Combine(drinker, beer, pub) := LIKES ⋈ FREQUENTS

All1(drinker, beer, pub, cost) := Combine ⋈ SERVES

//All2 is All1 with the fields renamed

All2(drinker2, beer2, pub2, cost2) := ρdrinker -> drinker2, beer -> beer2, pub -> pub2, cost -> cost2 (All1)

Q7 Contd.

//Same drinker, same pub, two different beers

Join(drinker, beer, pub, cost, drinker2, beer2, pub2, cost2) := All1 ⋈All1.drinker = All2.drinker2 AND All1.pub = All2.pub2 AND All1.beer != All2.beer2 All2

//Select the expensive beers

Expensive(drinker, beer, pub, cost, drinker2, beer2, pub2, cost2) := σcost > 3 OR cost2 > 3 (Join)

//Project the drinkers

Answer(drinker) := πdrinker (Expensive)