Ceng 302
Database Management Systems

Relational Model and Algebra

Prof. Dr. Adnan YAZICI
Department of Computer Engineering,
Middle East Technical University
Ankara

(Fall 2021)

Relational Data Model

- Structure of Relational Data Model
- Constraints of the model (keys)
- Fundamental Relational-Algebra-Operations
- Additional Relational-Algebra-Operations
- Extended Relational-Algebra-Operations
- Null Values

Relational Database: Definitions

- **Relational database**: a set of *relations*
- **Relation:** made up of 2 parts:
  - **Instance** : a *table*, with *rows* and *columns*. #Rows = *cardinality*, #fields = *degree (arity)*.
  - **Schema** : specifies name of relation, plus name and type of each column.

    E.g. **Students** (**sid**: string, **name**: string, **login**: string, **age**: integer, **gpa**: real).
- We can think of a relation as a *set* of *rows* or *tuples* (i.e., all rows are distinct).

Basic Structure

- Formally, given sets D_1, D_2, …. D_n a **relation** *r* is a subset of

  D_1 x D_2 x … x D_n

  Thus, a relation is a set of *n*-tuples (a_1, a_2, …, a_n) where each $a_i \in D_i$

- Example: If
  - *customer_name* = {Jones, Smith, Curry, Lindsay,…} /*set of all customers*/
  - *customer_street* = {Main, North, Park, …} /* set of all street names */
  - *customer_city* = {Harrison, Rye, Pittsfield, …} /*set of all city names*/

  Then *r* = {(Jones, Main, Harrison), (Smith, North, Rye), (Curry, North, Rye), (Lindsay, Park, Pittsfield)} is a relation over

  customer_name* x *customer_street* x *customer_city

Attribute Types

- Each attribute of a relation has a name
- The set of allowed values for each attribute is called the **domain** of the attribute
- Attribute values are (normally) required to be **atomic**; that is, indivisible
  - E.g. the value of an attribute can be an account number, but cannot be a set of account numbers
- Domain is said to be atomic if all its members are atomic
- The special value *null* is a member of every domain
- The null value causes complications in the definition of many operations

Relation Schema

- $A_1, A_2, \ldots, A_n$ are *attributes*

- $R = (A_1, A_2, \ldots, A_n)$ is a *relation schema*

  Example:

  Customer_schema = (*customer_name, customer_street, customer_city*)

- *r*(*R*) denotes a *relation r* on the *relation schema R*

  Example:

  customer (Customer_schema)

Relation Instance

- The current values (*relation instance*) of a relation are specified by a table
- An element *t* of *r* is a *tuple*, represented by a *row* in a table

Customer

attributes
(or columns)

customer_name	***customer_street***	**customer_city**
Jones	**Main**	**Harrison**
Smith	**North**	**Rye**
Curry	**North**	**Rye**
Lindsay	**Park**	**Pittsfield**

tuples
(or rows)

Example Instance of Students Relation

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

- Cardinality = 3, degree = 5, all rows distinct
- Do all columns in a relation instance have to be distinct?

Database

- A database consists of multiple relations
- Information about an enterprise is broken up into parts, with each relation storing one part of the information

  account : stores information about accounts
  depositor : stores information about which customer owns which account
  customer : stores information about customers

- Storing all information as a single relation such as
  ***bank* (*account_number, balance, customer_name*, ..)** results in
  - repetition of information
  - the need for null values
- **Normalization theory** deals with how to design relational schemas

Relational Query Languages

- A major strength of the relational model:
  - supports simple, powerful *querying* of data.

- Queries can be written intuitively, and the DBMS is responsible for efficient evaluation.
  - Precise semantics for relational queries.
  - Allows the optimizer to extensively re-order operations, and still ensure that the answer does not change.

The SQL Query Language

- Developed by IBM (system R) in the 1970s
- Need for a standard since it is used by many vendors
- Standards:
  - SQL-86
  - SQL-89 (minor revision)
  - SQL-92 (major revision, current standard)
  - SQL-99 (major extensions)

Creating Relations in SQL

- Creates the Students relation. Observe that the type (domain) of each field is specified and enforced by the DBMS whenever tuples are added or modified.

```
CREATE TABLE Students
    (sid: CHAR(20),
     name: CHAR(20),
     login: CHAR(10),
     age: INTEGER,
     gpa: REAL)
```

- As another example, the Enrolled table holds information about courses that students take.

```
CREATE TABLE Enrolled
    (sid: CHAR(20),
     cid: CHAR(20),
     grade: CHAR(2))
```

Adding and Deleting Tuples

- Can insert a single tuple using:

```
INSERT INTO Students (sid, name, login, age, gpa)
VALUES (53688, 'Smith', 'smith@ee', 18, 3.2)
```

- Can delete all tuples satisfying some condition (e.g., name = Smith):

```
DELETE
FROM Students S
WHERE S.name = 'Smith'
```

* *Powerful variants of these commands are available.*

Modifying Tuples

- Can modify the column values in an existing row using:

```
UPDATE  Students S
SET     S.age = S.age + 1, S.gpa = S.gpa –1
WHERE   S.sid = 53688

UPDATE Students S
SET    S.gpa = S.gpa – 0.1
WHERE  S.gpa >= 3.3
```

Integrity Constraints (ICs)

- IC: condition that must be true for *any* instance of the database; e.g., *domain constraints.*
  - ICs are specified when schema is defined.
  - ICs are checked when relations are modified.
- A *legal* instance of a relation is one that satisfies all specified ICs.
  - DBMS should not allow illegal instances.
- If the DBMS checks ICs, stored data is more faithful to real-world meaning.
  - IC Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.
  - ICs are used to guard against accidental damage to the database.
  - ICs mainly avoid data entry errors.

Keys

- Let $K \subseteq R$
- *K* is a **superkey** of *R* if values for *K* are sufficient to identify a unique tuple of each relation *r(R)*.
  - **Example**: {*customer_name, customer_street*} and {*customer_name*} are both superkeys of the *Customer* relation, if no two customers can possibly have the same name.
  - In real life, an attribute such as *customer_id* would be used instead of *customer_name* to uniquely identify customers.

Keys (Cont.)

- *K* is a **candidate key** if *K* is minimal
  Example: {*customer_name*} is a candidate key for *Customer*, since it is a superkey and no subset of it is a superkey.
- **Primary key:** a candidate key is chosen as the principal means of identifying tuples within a relation
  - Should we choose an attribute whose value never, or very rarely, changes.
  - E.g. *email address* is unique, but it may change

Examples:

- ***sid*** is a *key* for Students. (What about *name*?)
- ***email*** is also a candidate key for students.
- The set {***sid, gpa***} is a *superkey*.

Primary and Candidate Keys in SQL

- Possibly many candidate keys (specified using UNIQUE), one of which is chosen as the *primary key*.

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

sid	cid	grade
53666	COP4	A
53666	COP4	B-
53666	CDA3	A

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

Primary and Candidate Keys in SQL

- Possibly many *candidate keys* (specified using UNIQUE), one of which is chosen as the *primary key*.

- "Students can take only one course and receive a single grade for that course; further, no two students in a course receive the same grade."

sid	cid	grade
53666	COP4	A
53666	CDA3	B-
53444	COP4	A

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

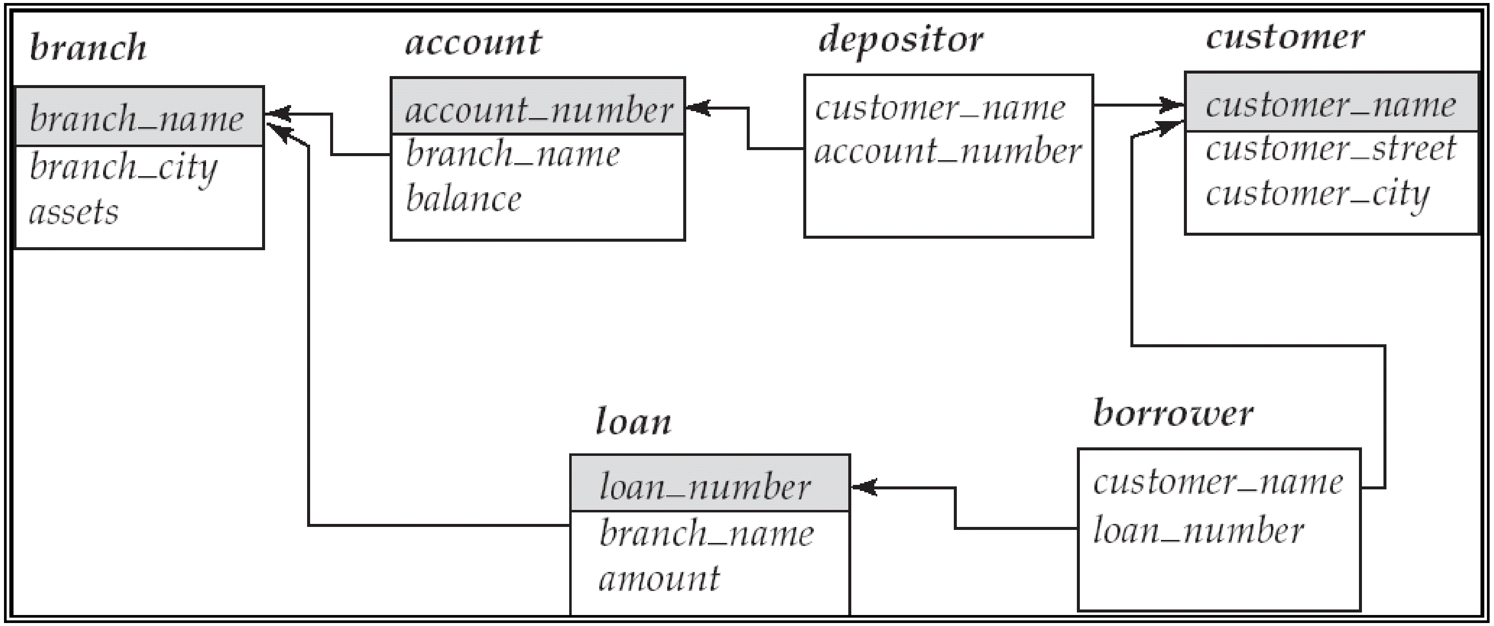
Foreign Keys, Referential Integrity

- *Foreign Key* (FK): Set of fields in one relation that is used to `refer' to a tuple in another relation. (FK must correspond to primary key of the second relation.) Like a `logical pointer'.

- E.g. *sid* in the *Enrolled* relation is a foreign key referring to *Students*:
  - Enrolled(***sid***: string, ***cid***: string, ***grade***: string)
  - If all foreign key constraints are enforced, *referential integrity* is achieved, i.e., no dangling references.

Foreign Keys

- A relation schema may have an attribute (X) that corresponds to the **primary key** of another relation. This attribute X is called a **foreign key**.
  - E.g. *customer_name* and *account_number* attributes of *depositor* are foreign keys to *customer* and *account* respectively.
  - Only values occurring in the primary key attribute of the **referenced relation** (*customer*) may occur as the foreign key attribute of the **referencing relation (***depositor***).**
- Schema diagram



Foreign Keys in SQL

- Only students listed in the Students relation should be allowed to enroll for courses. This is called the **referencial integrity constraint.**

```
CREATE TABLE Enrolled
  (sid CHAR(20), cid CHAR(20), grade CHAR(2),
   PRIMARY KEY (sid,cid),
   FOREIGN KEY (sid) REFERENCES Students )
```

Enrolled

sid	cid	grade
53666	Carnatic101	C
53666	Reggae203	B
53650	Topology112	A
53666	History105	B

Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Enforcing Referential Integrity

- Consider *Students* and *Enrolled*; ***sid*** in *Enrolled* is a foreign key that references *Students*.
- What should be done if an *Enrolled* tuple with a non-existent ***student id*** is inserted? (*Reject it!*)
- What should be done if a *Students* tuple is deleted?
  - Also delete all *Enrolled* tuples that refer to it.
  - Disallow deletion of a *Students* tuple that is referred to.
  - Set ***sid*** in *Enrolled* tuples that refer to it to a ***default sid***.
  - (In SQL, also: Set ***sid*** in *Enrolled* tuples that refer to it to a special value *null*, denoting *`unknown'* or *`inapplicable'*.)
- Similar if primary key of *Students* tuple is updated.

Referential Integrity in SQL/92

- SQL/92 supports all 4 options on deletes and updates.
  - Default is NO ACTION (*delete/update is rejected*)
  - CASCADE (also delete all tuples that refer to deleted tuple)
  - SET NULL / SET DEFAULT (sets foreign key value of referencing tuple)

```
CREATE TABLE Enrolled
  (sid CHAR(20) DEFAULT '9999',
   cid CHAR(20),
   grade CHAR(2),
   PRIMARY KEY (sid,cid),
   FOREIGN KEY (sid)
     REFERENCES Students
          ON DELETE CASCADE
          ON UPDATE NO ACTION )
```

Query Languages

- Language in which user requests information from the database.
- Categories of languages
  - Procedural
  - Non-procedural, or declarative
- “Pure” languages:
  - Relational algebra
  - Tuple relational calculus
  - Domain relational calculus
- Pure languages form underlying basis of query languages that people use.

Relational Algebra

- The Relational Algebra is procedural; you tell it how to construct the result
- It consists of a set of **operators** which, when applied to relations, yield relations (closed algebra)

$R \cup S$	**union**, R UNION S
$R \cap S$	**intersection**, R INTERSECT S
$R \setminus S$	**set difference**, R MINUS S
$R \times S$	**Cartesian product**, R JOIN S (no shared attributes)
$\pi_{A1, A2, ..., An}(R)$	**projection**, R[A1, A2, ..., An]
$\sigma_{expression}(R)$	**selection**, R WHERE EXPRESSION
$R \bowtie S$	**natural join**, R JOIN S (no shared attributes)
$R \bowtie_{\theta} S$	**theta-join**, via selection from $\times$
$R \div S$	**divideby**, R DIVIDEBY S
ρ [A1 B1,.., An Bn]	**rename**, R[A1 B1,.., An Bn]

Relational Algebra

- **Basic operations:**
  - *Selection* (σ) Selects a subset of rows from relation.
  - *Projection* (π) Deletes unwanted columns from relation.
  - *Cross-product* ($\times$) Allows us to combine two relations.
  - *Set-difference* ($-$) Tuples in reln. 1, but not in reln. 2.
  - *Union* ($\cup$) Tuples in reln. 1 and in reln. 2.

  or

  - *Intersect* ($\cap$) Tuples in reln. 1 and in reln. 2.
- **Additional operations:**
  - Intersection (or union), *join*, division, renaming: Not essential, but (very!) useful.
- Since each operation returns a relation, operations can be *composed*! (Algebra is "closed".)

Select Operation – Example

Relation r:

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

$\sigma_{A=B \wedge D > 5}(r)$:

A	B	C	D
α	α	1	7
β	β	23	10

Select Operation

- Notation: $\sigma_p(r)$
- p is called the **selection predicate**
- Defined as:

$$\sigma_p(\boldsymbol{r}) = \{t \mid t \in r \textbf{ and } p(t)\}$$

Where p is a formula in propositional calculus consisting of **terms** connected by $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)

Each **term** is one of:

<attribute> *op* <attribute> or <constant>

where ***op*** is one of: $=, \neq, >, \geq. <. \leq$

- Example of selection:

$$\sigma_{branch\_name = \text{“Perryridge”}}(account)$$

Project Operation – Example

Relation *r*:

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

$\Pi_{A,C}$ (*r*):

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

Project Operation

- Notation:

$$\prod_{A_1, A_2, \ldots, A_k} (r)$$

  where A_1, A_2 are attribute names and r is a relation name.
- The result is defined as the relation of k columns obtained by erasing the columns that are not listed
- Duplicate rows removed from result, since relations are sets
- **Example**: To eliminate the *branch_name* attribute of *account*

$$\prod_{account\_number,\ balance} (account)$$

Union Operation – Example

Relations *r, s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

r ∪ s:

A	B
α	1
α	2
β	1
β	3

Union Operation

- Notation: $r \cup s$
- Defined as:

$$r \cup s = \{t \mid t \in r \text{ or } t \in s\}$$

- For $r \cup s$ to be valid, they must be **union compatiple,** which means;

1. *r, s* must have the *same* **arity** (same number of attributes)
2. The attribute domains must be **compatible**

Example: 2nd column of *r* deals with the same type of values as does the 2nd column of *s*)

Example: to find all customers with either an account or a loan

$$\Pi_{customer\_name}\,(depositor) \;\cup\; \Pi_{customer\_name}\,(borrower)$$

Set-Intersection Operation – Example

Relation *r, s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

$r \cap s$

A	B
α	2

Notation: $r \cap s$

Defined as:

$r \cap s = \{ \ t \mid t \in r \text{ and } t \in s \ \}$

Assume:

r, s are **union compatible**

Note: $r \cap s = r - (r - s)$

Set Difference Operation – Example

Relations *r*, *s*:

A	*B*
α	1
α	2
β	1

r

A	*B*
α	2
β	3

s

***r* – *s*:**

A	*B*
α	1
β	1

Set Difference Operation

- Notation $r - s$
- Defined as:

$$r - s = \{t \mid t \in r \textbf{ and } t \notin s\}$$

- Set differences must also be **union compatible** relations.

- As might noticed, **Difference**, **Union** and **Intersect** operations must be ***union-compatible;***
  - Same number of fields.
  - Attribute domains of *r* and *s* must be compatible, that is \`Corresponding' fields have the same type.

Cartesian-Product Operation – Example

Relations *r, s*:

A	*B*
α	1
β	2

r

C	*D*	*E*
α	10	a
β	10	a
β	20	b
γ	10	b

s

***r* x *s*:**

A	*B*	*C*	*D*	*E*
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Cartesian-Product Operation

- Notation r x s
- Defined as:

$$r \times s = \{t\,q \mid t \in r \textbf{ and } q \in s\}$$

- Assume that attributes of r(R) and s(S) are disjoint. That is,

$$R \cap S = \emptyset.$$

- If attributes of *r(R)* and *s(S*) are not disjoint, then renaming must be used.

Composition of Operations

- Can build expressions using multiple operations
- Example: $\sigma_{A=C}(r \times s)$
- *r x s*

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

- $\sigma_{A=C}(r \times s)$

A	B	C	D	E
α	1	α	10	a
β	2	β	10	a
β	2	β	20	b

Natural-Join Operation

Notation: $r \bowtie s$

- Let *r* and *s* be relations on schemas *R* and *S* respectively. Then, $r \bowtie s$ is a relation on schema $R \cup S$ obtained as follows:
  - Consider each pair of tuples t_r from *r* and t_s from *s*.
  - If t_r and t_s have the same value on each of the attributes in $R \cap S$, add a tuple *t* to the result, where
    - *t* has the same value as t_r on *r*
    - *t* has the same value as t_s on *s*
- Example:

  $R = (A, B, C, D)$

  $S = (E, B, D)$

  - Result schema = (A, B, C, D, E)
  - $r \bowtie s$ is defined as:

$$\Pi_{r.A,\ r.B,\ r.C,\ r.D,\ s.E}\ (\sigma_{r.B = s.B\ \wedge\ r.D = s.D}\ (r \times s))$$

Natural Join Operation – Example

- Relations r, s:

A	*B*	*C*	*D*
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	*D*	*E*
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	$\in$

s

r $\bowtie$ s

A	*B*	*C*	*D*	*E*
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

$$\Pi_{r.A,\ r.B,\ r.C,\ r.D,\ s.E}\ (\sigma_{r.B\ =\ s.B\ \wedge\ r.D\ =\ s.D}\ (r\ \text{x}\ s))$$

Rename Operation

- Allows us to name, and therefore to refer to, the results of relational-algebra expressions.
- Allows us to refer to a relation by more than one name.
- Example:

$$\rho_X(E)$$

returns the expression E under the name X

- If a relational-algebra expression E has arity n, then

$$\rho_{x(A_1, A_2, \ldots, A_n)}(E)$$

returns the result of expression E under the name X, and with the attributes renamed to A_1, A_2, …., A_n.

Division Operation

- Notation: $r \div s$
- Suited to queries that include the phrase "**for all**".
- Let r and s be relations on schemas R and S respectively where
  - $R = (A_1, \ldots, A_m, B_1, \ldots, B_n)$
  - $S = (B_1, \ldots, B_n)$

  The result of **r ÷ s** is a relation on schema

  $$R - S = (A_1, \ldots, A_m)$$

  $$r \div s = \{\, t \mid t \in \prod_{R\text{-}S}(r) \land \forall\, u \in s\, (\, tu \in r\,)\, \}$$

  where ***tu*** means the concatenation of tuples ***t*** and ***u*** to produce a single tuple.

  - **r ÷ s** contains all **t** tuples such that for every **u** tuple in **s**, there is a **tu** tuple in **r**. Or if the set of **u** values associated with a **t** value in **r** contains all **u** values in **s**, the **t** value is in **r ÷ s**.

Division Operation – Example

Relations *r, s*:

A	*B*
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
$\in$	6
$\in$	1
β	2

r

B
1
2

s

$$R - S = (A_1, \ldots, A_m)$$

$$r \div s = \{\, t \mid t \in \Pi_{R\text{-}S}(r) \wedge \forall u \in s\,(tu \in r)\,\}$$

***r* ÷ *s*:**

A
α
β

Examples of Division R/Si

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

R

pno
p2

S_1

pno
p2
p4

S_2

pno
p1
p2
p4

S_3

sno
s1
s2
s3
s4

R/S_1

sno
s1
s4

R/S_2

sno
s1

R/S_3

Another Division Example

Relations *r, s*:

A	*B*	*C*	*D*	*E*
α	a	α	a	*1*
α	a	γ	a	*1*
α	a	γ	b	*1*
β	a	γ	a	*1*
β	a	γ	b	*3*
γ	a	γ	a	*1*
γ	a	γ	b	*1*
γ	a	β	b	*1*

r

D	*E*
a	*1*
b	*1*

s

***r* ÷ *s*:**

A	*B*	*C*
α	a	γ
γ	a	γ

$$R - S = (A_1, \ldots, A_m)$$

$$r \div s = \{\, t \mid t \in \Pi_{R\text{-}S}(r) \wedge \forall u \in s\, (tu \in r) \,\}$$

Division Operation (Cont.)

Property

- Let $q = r \div s$
- Then q is the largest relation satisfying $q \times s \subseteq r$

- Definition in terms of the basic algebra operation

  Let $r(R)$ and $s(S)$ be relations, and let $S \subseteq R$

$$r \div s = \prod_{R-S}(r) - \prod_{R-S}((\prod_{R-S}(r) \times s) - \prod_{R-S,S}(r))$$

- $\prod_{R-S,S}(r)$ simply reorders attributes of r

- $\prod_{R-S}(\prod_{R-S}(r) \times s) - \prod_{R-S,S}(r))$ gives those tuples t in $\prod_{R-S}(r)$ such that for some tuple $u \in s$, $tu \notin r$.

Division Operation (Cont.)

$$r \div s = \Pi_{R-S}(r) - \Pi_{R-S}[(\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r)]$$

- $\Pi_{R-S,S}(r)$ simply reorders attributes of r
- $\Pi_{R-S}(\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$ gives those tuples t in $\Pi_{R-S}(r)$ such that for some tuple $u \in s$, $tu \notin r$.

$\Pi_{R-S}(r) - \Pi_{R-S}[(\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r)] = r \div s$

A
α
α
α
β
γ
δ
δ
δ
∈
∈
β

$\Pi_{R-S}(r)$

A	B
α	1
α	2
β	1
β	2
γ	1
γ	2
δ	1
δ	2
∈	1
∈	2

$\Pi_{R-S}(r) \times s)$

A	B
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
∈	6
∈	1
β	2

$\Pi_{R-S,S}(r)$

A
γ
δ
∈

$\Pi_{R-S}(\ldots)$

A
α
β

Example Instances

Reserves

sid	bid	day
22	101	10/10/96
22	103	12/10/96
22	102	13/10/96

Sailors

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

- We will use these instances of the Sailors, Boats and Reserves relations in our examples.

Boats

bid	bname	color
101	Intertake	blue
102	Intertake	red
103	Clipper	green

***Query:* "Find sailors who've reserved all boats."**

Division in SQL

Query: Find sailors who've reserved **all** boats.

```
SELECT  S.sname
FROM    Sailors S
WHERE   NOT EXISTS
           ((SELECT  B.bid
             FROM    Boats B)
            EXCEPT
            (SELECT  R.bid
             FROM    Reserves R
             WHERE   R.sid=S.sid))
```

- Let's do it without EXCEPT:

```
SELECT  S.sname
FROM    Sailors S
WHERE   NOT EXISTS (SELECT  B.bid
                    FROM    Boats B
                    WHERE   NOT EXISTS (SELECT  R.bid
                                        FROM    Reserves R
                                        WHERE   R.bid=B.bid
                                           AND  R.sid=S.sid))
```

Sailors S such that ...

there is no boat B without

a Reserves tuple showing S reserved B

or "Select each sailor such that there does not exist a boat that the sailor does not reserve it."

Querying using Relational Algebra

Query: Find names of sailors who've reserved a red boat

- Information about boat color only available in Boats; so need an extra join:

$$\pi_{sname}((\sigma_{color='red'} Boats) \bowtie \text{Reserves} \bowtie Sailors)$$

- Alternative solution:

$$\pi_{sname}(\pi_{sid}((\pi_{bid} \sigma_{color='red'} Boats) \bowtie \text{Res}) \bowtie Sailors)$$

A query optimizer can find this, given the first solution!

Assignment Operation

- The assignment operation ($\leftarrow$) provides a convenient way to express complex queries.
  - Write query as a sequential program consisting of
    - a series of assignments
    - followed by an expression whose value is displayed as a result of the query.
  - Assignment must always be made to a temporary relation variable.

Example: Write $r \div s$ as

$$temp1 \leftarrow \prod_{R-S} (r)$$

$$temp2 \leftarrow \prod_{R-S} ((temp1 \times s) - \prod_{R-S,S} (r))$$

$$result = temp1 - temp2$$

- The result to the right of the $\leftarrow$ is assigned to the relation variable on the left of the $\leftarrow$.
- May use variable in subsequent expressions.

Querying using Relational Algebra

Query: Find names of sailors who've reserved boat #103

- Solution 1: $\pi_{sname}((\sigma_{bid=103} \text{Reserves}) \bowtie Sailors)$

- Solution 2: $\rho\ (Temp1,\ \sigma_{bid=103} \text{Reserves})$

  $\rho\ (Temp2,\ Temp1 \bowtie Sailors)$

  $\pi_{sname}(Temp2)$

- Solution 3: $\pi_{sname}(\sigma_{bid=103}(\text{Reserves} \bowtie Sailors))$

Aggregate Functions and Operations

- **Aggregation function** takes a collection of values and returns a single value as a result.

  avg: average value
  min: minimum value
  max: maximum value
  sum: sum of values
  count: number of values

- **Aggregate operation** in relational algebra

$$_{G_1, G_2, \ldots, G_n}\vartheta_{F_1(A_1), F_2(A_2, \ldots, F_n(A_n)}(E)$$

E is any relational-algebra expression

- $G_1, G_2 \ldots, G_n$ is a list of attributes on which to group (can be empty)
- Each F_i is an aggregate function
- Each A_i is an attribute name

Aggregate Operation – Example

Relation *r*:

A	*B*	*C*
α	α	7
α	β	7
β	β	3
β	β	10

$\mathcal{g}_{\text{sum(c)}}\ (\text{r})$

sum(*C*)
27

Aggregate Operation – Example

Relation ***account*** grouped by ***branch-name***:

branch_name	*account_number*	*balance*
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

$$_{branch\_name}\,\mathcal{G}\,_{\mathbf{sum}(balance)}\,(account)$$

branch_name	sum(*balance*)
Perryridge	1300
Brighton	1500
Redwood	700

Aggregate Functions (Cont.)

Result of aggregation does not have a name

- Can use rename operation to give it a name
- For convenience, we permit renaming as part of aggregate operation

$$_{branch\_name}\,\mathcal{G}\,_{\mathbf{sum}(balance)\ \mathbf{as}\ sum\_balance}\,(account)$$

Outer Join

- An extension of the **join** operation that avoids loss of information.
- Computes the join and then adds tuples form one relation that does not match tuples in the other relation to the result of the join.
- Uses *null* values:
  - *null* signifies that the value is **unknown** or **does not exist**
  - All comparisons involving ***null*** are (roughly speaking) **false** by definition.
    - We shall study precise meaning of comparisons with nulls later

Outer Join – Example

Relation ***loan***

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

Relation ***borrower***

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

Outer Join – Example

loan

loan_number	*branch_name*	*amount*
L-170	**Downtown**	**3000**
L-230	**Redwood**	**4000**
L-260	**Perryridge**	**1700**

borrower

customer_name	*loan_number*
Jones	**L-170**
Smith	**L-230**
Hayes	**L-155**

Join

loan $\bowtie$ *borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	**Downtown**	**3000**	**Jones**
L-230	**Redwood**	**4000**	**Smith**

Left Outer Join

loan $⟕$ *borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	**Downtown**	**3000**	**Jones**
L-230	**Redwood**	**4000**	**Smith**
L-260	**Perryridge**	**1700**	***null***

Outer Join – Example

loan

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

borrower

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

Right Outer Join

loan ⟖ *borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	*null*	*null*	Hayes

Full Outer Join

loan ⟗ *borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*
L-155	*null*	*null*	Hayes

Relational Model - Operations

Powerful set-oriented query languages:

- Relational Algebra: procedural; describes how to compute a query

  Operators; Union, Select, Project, Cartesian Product,
  Difference, and

  Intersect, Join, Division, Outer Join, Outer Union, etc.

- Relational Calculus: declarative; describes the desired result,
  Insert, delete, and update capabilities

  e.g., SQL, QBE

Outer Union

Display all data values from Table df1 and table df4, but overlay common attributes.

df1

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

df4

	B	D	F
2	B2	D2	F2
3	B3	D3	F3
6	B6	D6	F6
7	B7	D7	F7

Result

	A	B	C	D	F
0	A0	B0	C0	D0	NaN
1	A1	B1	C1	D1	NaN
2	A2	B2	C2	D2	NaN
3	A3	B3	C3	D3	NaN
2	NaN	B2	NaN	D2	F2
3	NaN	B3	NaN	D3	F3
6	NaN	B6	NaN	D6	F6
7	NaN	B7	NaN	D7	F7

Outer Union

Table ONE

X	A
1	a
1	a
1	b
2	c
3	v
4	e
6	g

Table TWO

X	B
1	x
2	y
3	z
3	v
5	w

```
select *
   from one
outer union corr
select *
   from two;
```

Final Results

X	A	B
1	a	
1	a	
1	b	
2	c	
3	v	
4	e	
6	g	
1		x
2		y
3		z
3		v
5		w

Relational Calculus

- The Relational Calculus is non-procedural. It allows you to express a result relation using a predicate on tuple variables (tuple calculus):

$$\{ t \mid P(t) \}$$

  or on domain variables (domain calculus):

$$\{ <x_1, x_2, \ldots, x_n> \mid P(<x_1, x_2, \ldots, x_n>) \}$$

- You tell the system which result you want, but not how to construct it.

Relational Calculus

FLT-WEEKDAY

<u>flt#</u>	<u>weekday</u>

Query: Find FLT# for all flights scheduled for Mondays

Tuple calculus:

$\{t.FLT\# \mid FLT\text{-}WEEKDAY(t) \wedge t.WEEKDAY = MO\}$

Domain calculus:

$\{<FLT\#> \mid <FLT\#, WEEKDAY> \in FLT\text{-}WEEKDAY \wedge WEEKDAY = MO\}$

Relational Calculus

Tuple and domain calculus for join operation: $\{t \mid P(t)\}$

FLT-WEEKDAY

<u>flt#</u>	<u>weekday</u>

FLT-INSTANCE

<u>flt#</u>	<u>date</u>	plane#	#avail-seats

Query: Find and make a list with complete flight instance information

Tuple Calculus:

$$\{s.FLT\#,\ s.WEEKDAY,\ t.DATE,\ t.PLANE\#,\ t.\#AVAIL\text{-}SEATS \mid FLT\text{-}WEEKDAY(s) \wedge FLT\text{-}INSTANCE(t) \wedge s.FLT\# = t.FLT\# \}$$

Domain Calculus:

$$\{\langle FLT\#,\ WEEKDAY,\ DATE,\ PLANE\#,\ \#AVAIL\text{-}SEATS \rangle \mid \langle FLT\#,\ WEEKDAY\rangle \in FLT\text{-}WEEKDAY \wedge \langle FLT\#,\ DATE,\ PLANE\#,\ \#AVAIL\text{-}SEATS\rangle \in FLT\text{-}INSTANCE \wedge FLT\# = FLT\#\}$$

Null Values

- It is possible for tuples to have a null value, denoted by ***null***, for some of their attributes
- *null* signifies an **unknown** value or that a value **does not exist.**
- The result of any arithmetic expression involving null is ***null.***
- **Aggregate functions** simply ignore **null** values (as in SQL)
- For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same (as in SQL)

Null Values

- Comparisons with null values return the special truth value: ***unknown***
  - If *false* were used instead of *unknown*, then *not (A < 5)* would not be equivalent to *A >= 5*
- 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*
- In SQL "*P* **is unknown**" evaluates to true if predicate *P* evaluates to *unknown*
- Result of select predicate is treated as *false* if it evaluates to *unknown*