# Chapter 3

The Relational Model

Transparencies

## Chapter 3 - Objectives

- The origins of the relational model.
- The terminology of the relational model.
- How tables are used to represent data.
- The connection between mathematical relations and relations in the relational model.

## Chapter 3 - Objectives

- Properties of database relations.
- How to identify candidate, primary, and foreign keys.
- The meaning of entity integrity and referential integrity.
- The categories of relational Data Manipulation Languages (DMLs).

## Chapter 3 - Objectives

- How to form queries in relational algebra.
- The purpose and advantages of views in relational systems.

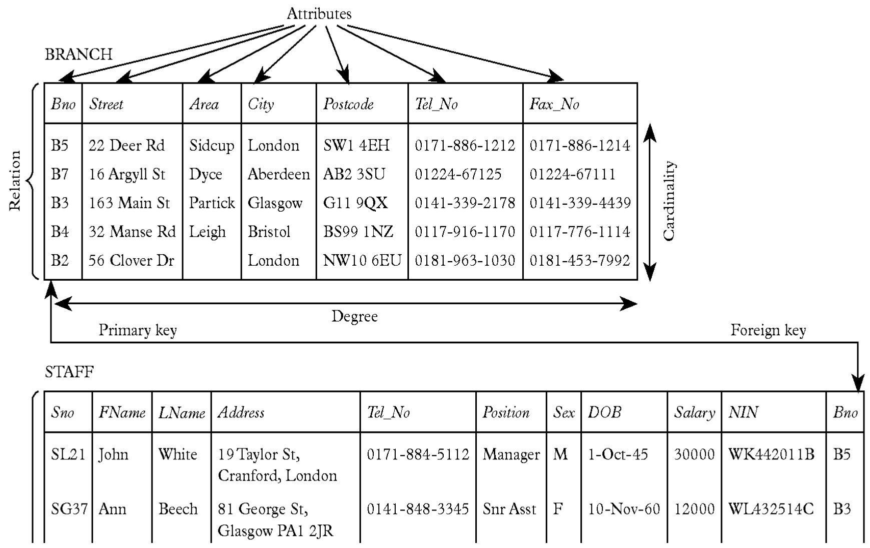
## Relational Model Terminology

- A relation is a table with columns and rows.
  - Only applies to logical structure of the database, not the physical structure.
- Attribute is a named column of a relation.
- Domain is a set of allowable values for one or more attributes.

## Relational Model Terminology

- Tuple is a row of a relation.
- Degree is a number of attributes in a relation.
- Cardinality is a number of tuples in a relation.
- Relational Database is a collection of normalized relations.

## Instances of Branch and Staff (part) Relations



BRANCH

| *Bno* | *Street* | *Area* | *City* | *Postcode* | *Tel_No* | *Fax_No* |
|---|---|---|---|---|---|---|
| B5 | 22 Deer Rd | Sidcup | London | SW1 4EH | 0171-886-1212 | 0171-886-1214 |
| B7 | 16 Argyll St | Dyce | Aberdeen | AB2 3SU | 01224-67125 | 01224-67111 |
| B3 | 163 Main St | Partick | Glasgow | G11 9QX | 0141-339-2178 | 0141-339-4439 |
| B4 | 32 Manse Rd | Leigh | Bristol | BS99 1NZ | 0117-916-1170 | 0117-776-1114 |
| B2 | 56 Clover Dr |  | London | NW10 6EU | 0181-963-1030 | 0181-453-7992 |

STAFF

| *Sno* | *FName* | *LName* | *Address* | *Tel_No* | *Position* | *Sex* | *DOB* | *Salary* | *NIN* | *Bno* |
|---|---|---|---|---|---|---|---|---|---|---|
| SL21 | John | White | 19 Taylor St, Cranford, London | 0171-884-5112 | Manager | M | 1-Oct-45 | 30000 | WK442011B | B5 |
| SG37 | Ann | Beech | 81 George St, Glasgow PA1 2JR | 0141-848-3345 | Snr Asst | F | 10-Nov-60 | 12000 | WL432514C | B3 |

## Examples of Attribute Domains

| *Attribute* | *Domain Name* | *Meaning* | *Domain Definition* |
|---|---|---|---|
| Bno | BRANCH_NUMBERS | The set of all possible branch numbers | character: size 3, range B1–B99 |
| Street | STREET_NAMES | The set of all street names in Britain | character: size 25 |
| Area | AREA_NAMES | The set of all area names in Britain | character: size 20 |
| City | CITY_NAMES | The set of all city names in Britain | character: size 15 |
| Pcode | POST_CODES | The set of all postcodes in Britain | character: size 8 |
| Tel_No | TELFAX_NUMBERS | The set of all telephone and fax numbers in Britain | character: size 13 |
| Fax_No | TELFAX_NUMBERS | The set of all telephone and fax numbers in Britain | character: size 13 |
| Sex | SEX | The sex of a person | character: size 1, value M or F |
| DOB | DATES_OF_BIRTH | Possible values of staff birth dates | date, range from 1-Jan-20, format dd-mmm-yy |
| Salary | SALARIES | Possible values of staff salaries | monetary: 7 digits, range 6000.00–40000.00 |

## Mathematical Relations

- **Mathematical definition of relation**
  - **Consider two sets, $D_1$ and $D_2$, where $D_1$ = {2, 4} and $D_2$ = {1, 3, 5}.**
  - **Cartesian product is $D_1$ ´ $D_2$, the set of all ordered pairs, first element is member of $D_1$ and second element is member of $D_2$.**
  - **Alternative way is to find all combinations of elements with first from $D_1$ and second from $D_2$.**

**$D_1$ ´ $D_2$ = {(2, 1), (2, 3), (2, 5), (4, 1), (4, 3), (4, 5)}**

## Mathematical Relation

- **Any subset of Cartesian product is a relation. For example**

  **$R$ = {(2, 1), (4, 1)}**

- **May specify which pairs are in relation using some condition for selection. For example, second element is 1**

  **$R$ = {($x$, $y$) | $x$ Î $D_1$, $y$ Î $D_2$, and $y$ = 1}**

## Mathematical Relations

- **Using same sets, form another relation, *S*, where first element is always twice the second.**

  **$S$ = {($x$, $y$) | $x$ Î $D_1$, $y$ Î $D_2$, and $x$ = 2$y$}**

- **Only one ordered pair in the Cartesian Product satisfies this condition.**

  **$S$ = {(2, 1)}**

## Mathematical Relations

- **Consider three sets $D_1$, $D_2$, and $D_3$ with Cartesian Product $D_1$ ´ $D_2$ ´ $D_3$. For example**

  **$D_1$ = {1, 3}   $D_2$ = {2, 4}   $D_3$ = {5, 6}**

  **$D_1$ ´ $D_2$ ´ $D_3$ = {(1,2,5), (1,2,6), (1,4,5), (1,4,6), (3,2,5), (3,2,6), (3,4,5), (3,4,6)}**

- **Any subset of these ordered triples is a relation.**

## Mathematical Relations

- To define a general relation on *n* domains…let $D_1$, $D_2, \ldots, D_n$ be *n* sets with Cartesian product defined as

  $$D_1 \times D_2 \times \ldots \times D_n = \{(d_1, d_2, \ldots, d_n) \mid d_1 \in D_1, d_2 \in D_2, \ldots, d_n \in D_n\}$$

  usually written as

  $$\overset{n}{\underset{i=1}{\mathrm{X}}} D_i$$

- In defining relations we specify the sets, or *domains*, from which we chose values.

## Properties of Relations

- Relation name is distinct from all other relations.
- Each cell of relation contains exactly one atomic (single) value.
- Each attribute has a distinct name.
- Values of an attribute are all from the same domain.
- Order of attributes has no significance.
- Each tuple is distinct; there are no duplicate tuples.
- Order of tuples has no significance, theoretically

## Relational Keys

- Superkey
  - An attribute or a set of attributes that uniquely identifies a tuple within a relation.

## Relational Keys

- Candidate Key
  - A superkey (K) such that no proper subset is a superkey within the relation.
  - In each tuple of R, the values of K uniquely identify that tuple (uniqueness).
  - No proper subset of K has the uniqueness property (irreducibility).

## Relational Keys

- Primary Key
  - Candidate key selected to identify tuples uniquely within  relation.
- Alternate Keys
  - Candidate keys not selected as the primary key.
- Foreign Key
  - An attribute or set of attributes within one relation that matches candidate key of some (possibly same) relation.

## Relational Integrity

- Null
  - Represents a value for an attribute that is currently unknown or is not applicable for this tuple.
  - Deals with incomplete or exceptional data.
  - Null represents the absence of a value and is not the same as zero or spaces, which are values.

## Relational Integrity

- Entity Integrity
  - In a base relation, no attribute of a primary key can be null.
- Referential Integrity
  - If foreign key exists in a relation, either the foreign key value must match a candidate key value of some tuple in its home relation or foreign key value must be wholly null.

## Relational Integrity

- Enterprise Constraints
  - Additional rules specified by users or database administrators.

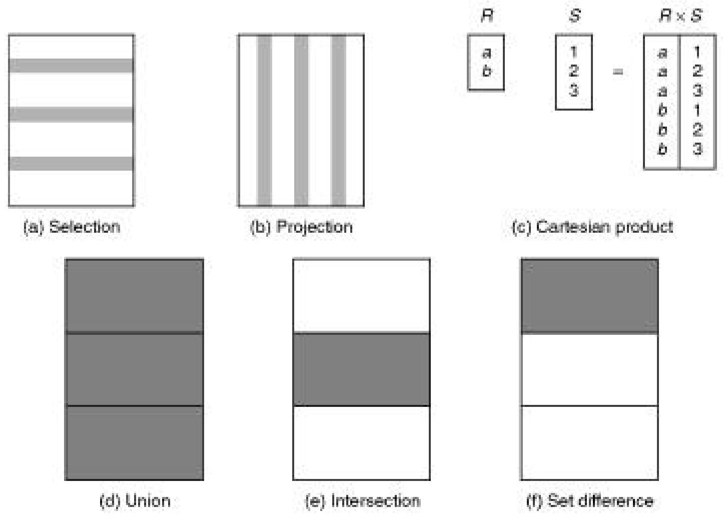
## Relational Algebra

- Relational algebra operations work on one or more relations to define another relation without changing the original relations.
- Thus, both operands and results are relations, so output from one operation can become input to another operation.
- This allows expressions to be nested, just as in arithmetic. This property is called closure.

## Relational Algebra

- There are 5 basic operations in relational algebra: Selection, Projection, Cartesian Product, Union, and Set Difference.
- Performs most of the data retrieval operations needed.
- Also have Join, Intersection, and Division operations that can be expressed in terms of 5 basic operations.

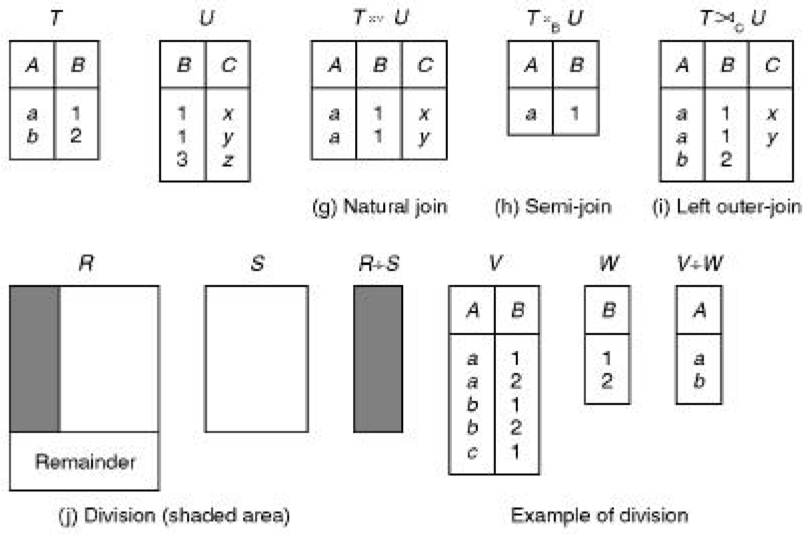
## Relational Algebra Operations



(a) Selection (b) Projection (c) Cartesian product

(d) Union (e) Intersection (f) Set difference

## Relational Algebra Operations



(g) Natural join (h) Semi-join (i) Left outer-join

(j) Division (shaded area) Example of division

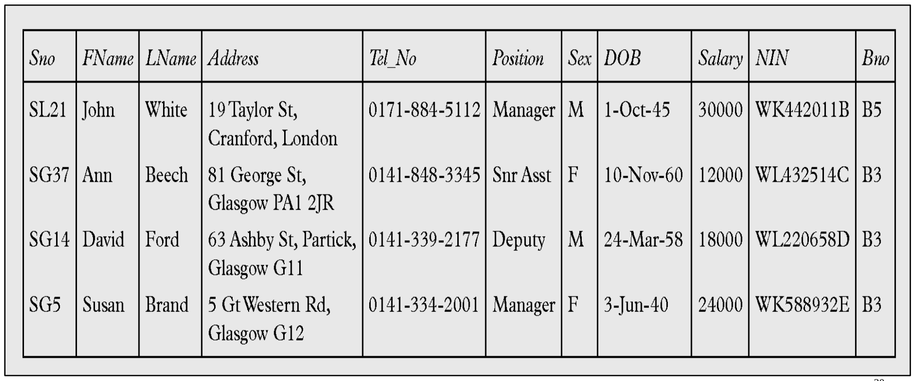
## Selection (or Restriction)

- $s_{\textbf{predicate}}$ **(R)**
  - **Selection operation works on a single relation R and defines a relation that contains only those tuples (rows) of R that satisfy the specified condition (*predicate*).**

## Example - Selection (or Restriction)

- **List all staff with a salary greater than £10,000.**

  $s_{\textbf{salary > 10000}}$ **(Staff)**



| *Sno* | *FName* | *LName* | *Address* | *Tel_No* | *Position* | *Sex* | *DOB* | *Salary* | *NIN* | *Bno* |
|---|---|---|---|---|---|---|---|---|---|---|
| SL21 | John | White | 19 Taylor St, Cranford, London | 0171-884-5112 | Manager | M | 1-Oct-45 | 30000 | WK442011B | B5 |
| SG37 | Ann | Beech | 81 George St, Glasgow PA1 2JR | 0141-848-3345 | Snr Asst | F | 10-Nov-60 | 12000 | WL432514C | B3 |
| SG14 | David | Ford | 63 Ashby St, Partick, Glasgow G11 | 0141-339-2177 | Deputy | M | 24-Mar-58 | 18000 | WL220658D | B3 |
| SG5 | Susan | Brand | 5 Gt Western Rd, Glasgow G12 | 0141-334-2001 | Manager | F | 3-Jun-40 | 24000 | WK588932E | B3 |

## Projection

- $P_{\textbf{col1, . . . , coln}}\textbf{(R)}$
  - **Projection operation works on a single relation R and defines a relation that contains a vertical subset of R, extracting the values of specified attributes and eliminating duplicates.**

## Example - Projection

- **Produce a list of salaries for all staff, showing only the Sno, FName, LName, and Salary details.**

$P_{\textbf{sno, fname, lname, salary}}\textbf{(Staff)}$

| *Sno* | *FName* | *LName* | *Salary* |
|---|---|---|---|
| SL21 | John | White | 30000 |
| SG37 | Ann | Beech | 12000 |
| SG14 | David | Ford | 18000 |
| SA9 | Mary | Howe | 9000 |
| SG5 | Susan | Brand | 24000 |
| SL41 | Julie | Lee | 9000 |

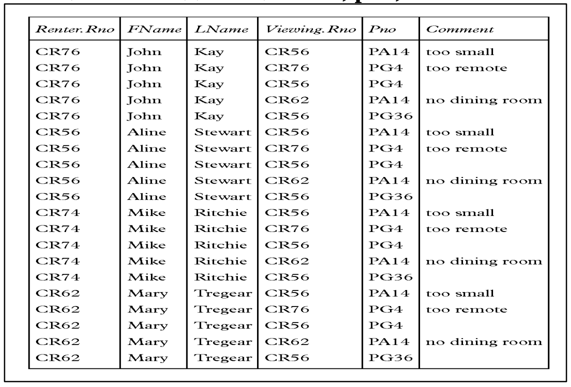
## Cartesian Product

- **R X S**
  - **The Cartesian product operation defines a relation that is the concatenation of every tuple of relation R with every tuple of relation S.**

## Example - Cartesian Product

- **List the names and comments of all renters who have viewed a property.**

**(P $_{\text{rno, fname, lname}}$(Renter)) X (P $_{\text{rno, pno,comment}}$ (Viewing))**



| *Renter.Rno* | *FName* | *LName* | *Viewing.Rno* | *Pno* | *Comment* |
|---|---|---|---|---|---|
| CR76 | John | Kay | CR56 | PA14 | too small |
| CR76 | John | Kay | CR76 | PG4 | too remote |
| CR76 | John | Kay | CR56 | PG4 | |
| CR76 | John | Kay | CR62 | PA14 | no dining room |
| CR76 | John | Kay | CR56 | PG36 | |
| CR56 | Aline | Stewart | CR56 | PA14 | too small |
| CR56 | Aline | Stewart | CR76 | PG4 | too remote |
| CR56 | Aline | Stewart | CR56 | PG4 | |
| CR56 | Aline | Stewart | CR62 | PA14 | no dining room |
| CR56 | Aline | Stewart | CR56 | PG36 | |
| CR74 | Mike | Ritchie | CR56 | PA14 | too small |
| CR74 | Mike | Ritchie | CR76 | PG4 | too remote |
| CR74 | Mike | Ritchie | CR56 | PG4 | |
| CR74 | Mike | Ritchie | CR62 | PA14 | no dining room |
| CR74 | Mike | Ritchie | CR56 | PG36 | |
| CR62 | Mary | Tregear | CR56 | PA14 | too small |
| CR62 | Mary | Tregear | CR76 | PG4 | too remote |
| CR62 | Mary | Tregear | CR56 | PG4 | |
| CR62 | Mary | Tregear | CR62 | PA14 | no dining room |
| CR62 | Mary | Tregear | CR56 | PG36 | |

## Example - Cartesian Product and Selection

- **Use selection operation to extract those tuples where Renter.Rno = Viewing.Rno.**

  $s_{\textbf{renter.rno = viewing.rno}}$**((**Õ$_{\textbf{rno,fname,lname}}$**(Renter))** C **(**Õ$_{\textbf{rno,pno,comment}}$**(Viewing)))**

| *Renter.Rno* | *FName* | *LName* | *Viewing.Rno* | *Pno* | *Comment* |
|---|---|---|---|---|---|
| CR76 | John | Kay | CR76 | PG4 | too remote |
| CR56 | Aline | Stewart | CR56 | PA14 | too small |
| CR56 | Aline | Stewart | CR56 | PG4 | |
| CR56 | Aline | Stewart | CR56 | PG36 | |
| CR62 | Mary | Tregear | CR62 | PA14 | no dining room |

- **Cartesian product and Selection can be reduced to a single operation called a *join*.**

## Union

- **R** È **S**
  - **Union of two relations R and S with I and J tuples, respectively, is obtained by concatenating them into one relation with a maximum of (I + J) tuples, duplicate tuples being eliminated. R and S must be union-compatible.**

## Example - Union

- **Construct a list of all areas where there is either a branch or a property.**

  P $_{\textbf{area}}$**(Branch)** È P$_{\textbf{area}}$ **(Property_for_Rent)**

| *Area* |
|---|
| Sidcup |
| Dyce |
| Partick |
| Leigh |
| Dee |
| Kilburn |
| Hyndland |

## Set Difference

- **R – S**
  - **The set difference operation defines a relation consisting of the tuples that are in relation R, but not in S. R and S must be union-compatible.**

## Example - Set Difference

- **Construct a list of all cities where there is a branch office but no properties.**

$P_{city}$ **(Branch) –** $P_{city}$ **(Property_for_Rent)**

| *City* |
|---|
| Bristol |

## Join Operations

- **Join is a derivative of Cartesian product.**
- **Equivalent to performing a selection, using the join predicate as the selection formula, over the Cartesian product of the two operand relations.**
- **One of the most difficult operations to implement efficiently in a relational DBMS and one of the reasons why relational systems have intrinsic performance problems.**

## Join Operations

- There are various forms of join operation
  - Theta-join
  - Equi-join (a particular type of theta-join)
  - Natural join
  - Outer join
  - Semi-join

## Theta-join (q-join)

- $R \bowtie_F S$
  - Defines a relation that contains tuples satisfying the predicate F from the Cartesian product of R and S.
  - The predicate F is of the form $R.a_i$ q $S.b_i$ where q may be one of the comparison operators (<, < =, >, > =, =, ~ =).

## Theta-join (θ-join)

- **We can rewrite the theta-join in terms of the basic Selection and Cartesian product operations.**

$$R \bowtie_F S = \sigma_F(R \times S)$$

- **Degree of a theta-join is sum of the degrees of the operand relations R and S. If predicate F contains only equality (=), the term *equi-join* is used.**

## Example - Equi-join

- **List the names and comments of all renters who have viewed a property.**

$$(\Pi_{rno,fname,lname}(Renter)) \bowtie_{renter.rno\ =\ viewing.rno} (\Pi_{rno,pno,comment}(Viewing))$$

| *Renter.Rno* | *FName* | *LName* | *Viewing.Rno* | *Pno* | *Comment* |
|---|---|---|---|---|---|
| CR76 | John | Kay | CR76 | PG4 | too remote |
| CR56 | Aline | Stewart | CR56 | PA14 | too small |
| CR56 | Aline | Stewart | CR56 | PG4 | |
| CR56 | Aline | Stewart | CR56 | PG36 | |
| CR62 | Mary | Tregear | CR62 | PA14 | no dining room |

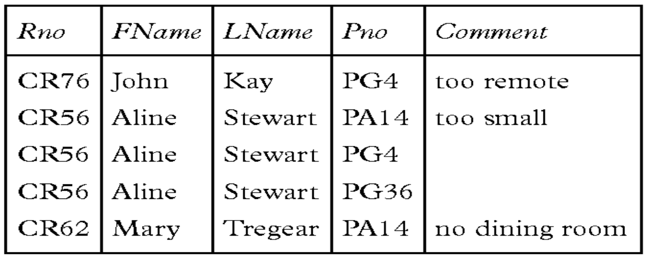
## Natural Join

- **R ⋈ S**
  - **Natural join is an equi-join of the two relations R and S over all common attributes x. One occurrence of each common attribute is eliminated from the result.**

## Example - Natural Join

- **List the names and comments of all renters who have viewed a property.**

**($P_{\mathbf{rno,fname,lname}}$ (Renter)) ⋈ ($P_{\mathbf{rno,pno,comment}}$(Viewing))**



| *Rno* | *FName* | *LName* | *Pno* | *Comment* |
|---|---|---|---|---|
| CR76 | John | Kay | PG4 | too remote |
| CR56 | Aline | Stewart | PA14 | too small |
| CR56 | Aline | Stewart | PG4 | |
| CR56 | Aline | Stewart | PG36 | |
| CR62 | Mary | Tregear | PA14 | no dining room |

## Outer Join

- Often in joining two relations, there is no matching value in the join columns. To display rows in the result that do not have matching values in the join column, we use the outer join.

- R ⟕ S
  - The (left) outer join is a join in which tuples from R that do not have matching values in the common columns of S are also included in the result relation.

## Example - Left Outer Join

- Produce a status report on property viewings.

$\Pi_{pno,street,city}$ (Property_for_Rent) ⟕ Viewing

| *Pno* | *Street* | *City* | *Rno* | *Date* | *Comment* |
|---|---|---|---|---|---|
| PA14 | 16 Holhead | Aberdeen | CR56 | 24-May-98 | too small |
| PA14 | 16 Holhead | Aberdeen | CR62 | 14-May-98 | no dining room |
| PL94 | 6 Argyll St | London | **null** | **null** | **null** |
| PG4 | 6 Lawrence St | Glasgow | CR76 | 20-Apr-98 | too remote |
| PG4 | 6 Lawrence St | Glasgow | CR56 | 26-May-98 | |
| PG36 | 2 Manor Rd | Glasgow | CR56 | 28-Apr-98 | |

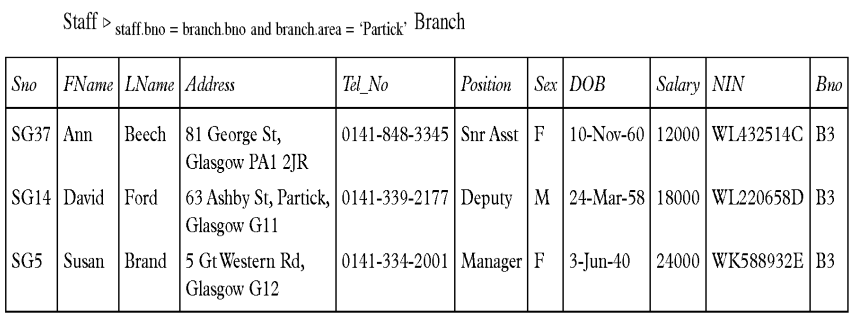
## Semi-join

- $R \triangleright_F S$
  - The semi-join operation defines a relation that contains the tuples of R that participate in the join of R with S.

## Example - Semi-join

- List complete details of all staff who work at the branch in Partick.

  Staff $_{\text{staff.bno = branch.bno and branch.area = 'Partick'}}$ Branch



Staff $\triangleright_{\text{staff.bno = branch.bno and branch.area = 'Partick'}}$ Branch

| *Sno* | *FName* | *LName* | *Address* | *Tel_No* | *Position* | *Sex* | *DOB* | *Salary* | *NIN* | *Bno* |
|---|---|---|---|---|---|---|---|---|---|---|
| SG37 | Ann | Beech | 81 George St, Glasgow PA1 2JR | 0141-848-3345 | Snr Asst | F | 10-Nov-60 | 12000 | WL432514C | B3 |
| SG14 | David | Ford | 63 Ashby St, Partick, Glasgow G11 | 0141-339-2177 | Deputy | M | 24-Mar-58 | 18000 | WL220658D | B3 |
| SG5 | Susan | Brand | 5 Gt Western Rd, Glasgow G12 | 0141-334-2001 | Manager | F | 3-Jun-40 | 24000 | WK588932E | B3 |

## Intersection

- **R Ç S**
  - **The intersection operation consists of the set of all tuples that are in both R and S.**
  - **R and S must be union-compatible.**

- **Expressed using basic operations**

  **R Ç S = R – (R – S)**

## Division

- **R ¸ S**
  - **The division operation consists of the set of tuples from R defined over the attributes C that match the combination of *every* tuple in S.**

- **Expressed using basic operations**

  $T_1 = \mathrm{P}_C(R)$

  $T_2 = \mathrm{P}_C((S \times T_1) – R)$

  $T = T_1 – T_2$

## Example - Division

- **Identify all renters who have viewed all properties with three rooms.**

**$(\text{P}_{rno,pno}\,(\text{Viewing}))$ ¸ $(\text{P}_{pno}\,(\text{s}_{rooms\,=\,3}\,(\text{Property\_for\_Rent})))$**

**$\Pi_{rno,pno}(\text{Viewing})$**

| *Rno* | *Pno* |
|---|---|
| CR56 | PA14 |
| CR76 | PG4 |
| CR56 | PG4 |
| CR62 | PA14 |
| CR56 | PG36 |

**$\Pi_{pno}(\sigma_{rooms\,=\,3}(\text{Property\_for\_Rent}))$**

| *Rno* |
|---|
| PG4 |
| PG36 |

**RESULT**

| *Rno* |
|---|
| CR56 |

## Relational Calculus

- **Relational calculus query specifies *what* is to be retrieved rather than *how* to retrieve it.**
  - **No description of how to evaluate a query.**
- **Based on a branch of symbolic logic called *predicate calculus*.**
- **When applied to databases, relational calculus is in two forms: *tuple-oriented* and *domain-oriented.***
- **In first-order logic or predicate calculus, a *predicate* is a truth-valued function with arguments.**

## Relational Calculus

- When we substitute values for the arguments, the function yields an expression, called a *proposition*, which can be either true or false.
- If a predicate contains a variable, as in ‘x is a member of staff’, there must be a range for x. When we substitute some values of this range for x, the proposition may be true; for other values, it may be false.

## Relational Calculus

- If P is a predicate, then we write the set of all x such that P is true for x, as

  **{x | P(x)}**

  **Predicates can be connected using Ù (AND), Ú (OR), and ~ (NOT)**

## Tuple-oriented Relational Calculus

- **Interested in finding tuples for which a predicate is true. Based on use of tuple variables.**
- **Tuple variable is a variable that 'ranges over' a named relation: that is, a variable whose only permitted values are tuples of the relation.**

## Tuple-oriented Relational Calculus

- **To specify the range of a tuple variable *S* as the Staff relation.**

  **RANGE OF S IS Staff**

- **To find the set of all tuples S such that P(S) is true.**

  **{S | P(S)}**

## Example - Tuple-oriented Relational Calculus

- To find the Sno, FName, LName, Address, Tel_No, Position, Sex, DOB, Salary, NIN, and Bno of all staff earning more than £10,000, we write

  RANGE OF S IS Staff

  {S | S.salary > 10000}

- S.salary means the value of the Salary attribute for the tuple S.

## Tuple-oriented Relational Calculus

- Universal quantifier is used in statements about every instance, such as:

  "  B (B.City ~ = ‘Paris’)

- Means ‘For all Branch tuples, the address is not in Paris’.

- Can also use ~ $ B (B.City = ‘Paris’) which means ‘There are no branches with an address in Paris’.

## Example - Tuple-oriented Relational Calculus

- List the names of all managers who earn more than £25,000.

  RANGE OF S IS Staff

  {S.fname, S.lname | S.position = ‘Manager’ Ù S.salary > 25000}

## Example - Tuple-oriented Relational Calculus

- List the names of staff who currently do not manage any properties.

  RANGE OF S IS Staff

  RANGE OF P IS Property_for_Rent

  {S.fname, S.lname | ~( $P (S.sno = P.sno) )}

- Can also use

  {S.fname, S.lname | " P (~(S.sno = P.sno) )}

## Domain-oriented Relational Calculus

- Uses variables that take values from domains instead of tuples of relations. If $P(d_1, d_2, \ldots, d_n)$ stands for a predicate with variables $d_1, d_2, \ldots, d_n$, then

  $$\{d_1, d_2, \ldots, d_n \mid P(d_1, d_2, \ldots, d_n)\}$$

- Means the set of all domain variables $d_1, d_2, \ldots, d_n$ for which the predicate, or formula, $P(d_1, d_2, \ldots, d_n)$ is true.

## Domain-oriented Relational Calculus

- We often test for a membership condition, to determine whether values belong to a relation.

- The expression $R(x, y)$ evaluates to true *if and only if* there is a tuple in relation $R$ with values $x, y$ for its two attributes.

## Example - Domain-oriented Relational Calculus

- Find the names of all managers who earn more than £25,000.

  **{fname, lname | $position, $salary**
  **(Staff (lname, position, salary) Ù position = ‘Manager’ Ù salary > 25000)}**

- Each attribute has a (variable) name. Condition Staff (lname, position, salary) ensures domain variables are restricted to attributes of same tuple.

## Example - Domain-oriented Relational Calculus

- List the staff who manage properties in Glasgow.

  **{fname, lname, pno | $sno Staff(sno, fname, lname) Ù $city (Property_for_Rent(pno, sno) Ù P.city = ‘Glasgow’)}**

## Domain-oriented Relational Calculus

- When domain relational calculus is restricted to safe expressions, it is equivalent to tuple relational calculus restricted to safe expressions, which is equivalent to relational algebra.

- Means every relational algebra expression has an equivalent relational calculus expression, and vice versa.

## Other Languages

- Transform-oriented languages are non-procedural languages that use relations to transform input data into required outputs (e.g. SQL).

- Graphical languages provide the user with a picture or illustration of the structure of the relation. The user fills in an example of what is wanted and the system returns the required data in that format (e.g  QBE).

## Other Languages

- **Fourth-generation languages (4GLs) can create a complete customized application using a limited set of commands in a user-friendly, often menu-driven environment.**
- **Some systems accept a form of *natural language*, sometimes called a fifth-generation language (5GL), although this development is still in its infancy.**

## Views

- **Base Relation**
  - **A named relation, corresponding to an entity in conceptual schema, whose tuples are physically stored in database.**
- **View**
  - **Dynamic result of one or more relational operations operating on the base relations to produce another relation.**

## Views

- A view is a virtual relation that does not actually exist in the database but is produced upon request, at time of request.
- Contents of a view are defined as a query on one or more base relations.
- Views are dynamic, meaning that changes made to base relations that affect view attributes are immediately reflected in the view.

## Purpose of Views

- Provides a powerful and flexible security mechanism by hiding parts of database from certain users.
- Permits users to access data in a customized way, so that same data can be seen by different users in different ways, at same time.
- It can simplify complex operations on base relations.

## Updating Views

- All updates to a base relation should be immediately reflected in all views that reference that base relation.
- If view is updated, underlying base relation should reflect change.

## Updating Views

- However, there are restrictions on types of modifications that can be made through views:
  - Updates are allowed if query involves a single base relation and contains a candidate key of base relation.
  - Updates are not allowed involving multiple base relations.
  - Updates are not allowed involving aggregation or grouping operations.

## Updating Views

- Classes of views are defined as theoretically not updateable, theoretically updateable and partially updateable.

## When is a DBMS Relational?

- In 1985, Codd specified 12 rules (13 with Rule 0, the foundational rule) for a relational DBMS.

- Rules in five functional areas
  - Foundational rules
  - Structural rules
  - Integrity rules
  - Data manipulation rules
  - Data independence rules