Learning Objectives:

- Explain the features of the Relational Model: Components of a relation, Properties of a relation, Null values, and Relational Operators
- Make use of a subset of the relational operators to solve data queries using symbolic notation: SELECT, PROJECT and JOIN
- Appreciate that the relational database model takes a logical view of data and how data redundancy is dealt with
- Describe the basic relational database components: Entities, Attributes, Relationships amongst entities, Integrity constraints, and Data Dictionary
- Describe the relational table’s components and characteristics and contrast the table with a relation
- Explain how keys are used in the relational database environment: candidate keys, primary keys, alternate keys, foreign keys, and secondary keys
- Describe the role of an index in the relational database model

References:

Where We Are

- Introduction to Database Systems
- The Relational Model
- Database Lifecycle
  - Conceptual Design
  - Logical Design
  - Physical Design
- Implementation
  - SQL (DML)
  - SQL (DDL & DCL)
  - Transaction Management
- Normalisation
- Database Administration
- Data Warehousing & Data Mining
Relational Model

• Dr. E. F. Codd proposed the relational model for database systems in 1970.
• It is the basis for the relational database management system (RDBMS).
• The relational model consists of the following:
  – Collection of objects or relations
    > domain, attribute, relation, tuple, primary key, degree, cardinality
  – Data integrity for accuracy and consistency
    > entity integrity and referential integrity
  – Set of operators to act on the relations
    > relational algebra, relational calculus
Relational Model Objects

• **Domain**
  - a set of atomic (indivisible) values
  - are pools of values from which the actual values appearing in attributes are drawn
    > specify
      - name
      - data type
      - data format

• **Examples:**
  - gender domain
    > one character string with allowable values of M, m, F, f
  - name domain
    > twenty character string
  - credit_limit domain
    > money in the range $1,000 to $99,999
Relational Model Objects

• Relation
  – a named set of attributes
  – consists of two parts: a heading and a body

• Relation Heading
  – also called a Relational Schema consists of a fixed set of attributes
    > R (A1,A2,.......An)
    – R = relation name, Ai = attribute i
  – each attribute corresponds to one underlying domain:
    > Customer relation heading:
      – CUSTOMER (custno, custname, custadd, credlimit)
        » dom(custno) = customer_number
        » dom(custname) = name
        » dom(custadd) = address
        » dom(credlimit) = credit_limit

<table>
<thead>
<tr>
<th>custno</th>
<th>custname</th>
<th>custadd</th>
<th>credlimit</th>
</tr>
</thead>
</table>


Relation

• **Relation Body**
  – Also called Relation Instance (state)
    > \( r(R) = \{t_1, t_2, t_3, \ldots, t_m\} \)
    > consists of a time-varying set of \( n \)-tuples
      – Relation \( R \) consists of tuples \( t_1, t_2, t_3 \ldots t_m \)
      – \( m \) = number of tuples = relation cardinality
    > each \( n \)-tuple is an ordered list of \( n \) values
    > \( t = < v_1, v_2, \ldots, v_n > \)
      – \( n \) = number of values in tuple = relation degree
  – In the tabular representation:
    > Relation heading \( \Rightarrow \) column headings
    > Relation body \( \Rightarrow \) set of data rows
### Relation

#### Relation Body
- tabular representation – data rows

<table>
<thead>
<tr>
<th>custno</th>
<th>custname</th>
<th>custadd</th>
<th>credlimit</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMI13</td>
<td>SMITH</td>
<td>Wide Rd, OnePlace, 1111</td>
<td>2000</td>
</tr>
<tr>
<td>JON44</td>
<td>JONES</td>
<td>Narrow St, Somewhere 2222</td>
<td>10000</td>
</tr>
<tr>
<td>BRO23</td>
<td>BROWN</td>
<td>Here Rd, Lost, 3333</td>
<td>10000</td>
</tr>
</tbody>
</table>

**Relation heading**

**Relation body**
Relation Properties

- **No duplicate tuples**
  - by definition sets do not contain duplicate elements
    > hence tuples are unique

- **Tuples are unordered within a relation**
  - by definition sets are not ordered
    > hence tuples can only be accessed by content

- **No ordering of attributes within a tuple**
  - by definition sets are not ordered

- **No multivalued attributes in a relation**
  - an entry at an intersection of each row and column is atomic (single-valued)
Relation

- The basic structure in Codd’s Model is the mathematical concept of a RELATION
- map relations to the 'concept' of a table
  - Relation - abstract object
  - Table - pictorial representation
  - Storage structure - "real thing" - eg. ISAM file
- Tabular representation
  - normally nothing to prevent duplicate rows
  - rows are ordered
  - columns are ordered
- Tables and relations are not the same 'things'
Logical View of Data

• **Relational Database**
  – Designer focuses on logical representation rather than physical
  – Use of table advantageous
    > Structural and data independence
    > Related records stored in independent tables
    > Logical simplicity
  – Allows for more effective design strategies

• **Entities and Attributes**
  – Entity is a person, place, event, or thing about which data is collected
  – Attributes are characteristics of the entity

• **Tables**
  – hold related entities or entity set
  – also called relations
  – comprise of rows and columns
Table Characteristics

- Two-dimensional structure with rows and columns
- Rows (tuples) represent a single entity
- Columns represent attributes
- Row/column intersection represent a single value
- Tables must have an attribute or collection of attributes to uniquely identify each row
- Column values all have same data format
- Each column has range of values called attribute domain
- Order of the rows and columns is immaterial to the DBMS
### Relational Model – Tabular Representation

<table>
<thead>
<tr>
<th>ROW or TUPLE</th>
<th>COLUMN or ATTRIBUTE</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMI13</td>
<td>SMITH</td>
<td>Wide Rd, OnePlace, 1111</td>
</tr>
<tr>
<td>JON44</td>
<td>JONES</td>
<td>Narrow St, Somewhere 2222</td>
</tr>
<tr>
<td>BRO23</td>
<td>BROWN</td>
<td>Here Rd, Lost 3333</td>
</tr>
<tr>
<td>JON16</td>
<td>JONES</td>
<td>Wide St, Nowhere, 3009</td>
</tr>
<tr>
<td>WHI05</td>
<td>WHITE</td>
<td>Fly Rd, Somewhere, 2980</td>
</tr>
<tr>
<td>GRE21</td>
<td>GREEN</td>
<td>Palace St, Berwick, 3806</td>
</tr>
</tbody>
</table>
Controlled Redundancy

- Makes the relational database work
- Tables within the database share common attributes that enable us to link tables together
- Multiple occurrences of values in a table are not redundant when they are required to make the relationship work
NULL

• **NULL is NOT a value**
  – is a representation of the fact that there is NO VALUE

• **Reasons for a NULL:**
  – **VALUE NOT APPLICABLE** -
    > EMP relation - empno, deptno, salary, commission
    – commission only applies to staff in sales dept
  – **VALUE UNKNOWN** -
    > Joe's salary is NULL, Joe’s salary is currently unknown
- **VALUE DOES NOT EXIST** -
  > Tax File Number - is applicable to all employees BUT Joe may not have a number at this time

- **VALUE UNDEFINED** -
  > Certain items explicitly undefined eg. divide by zero
    - Columns Number_of_payments Total_payments
    - Column Average_payment_made
    - If Number_of_payments = 0 => Average undefined
NULL

• Most SQL systems implement NULLs as VALUE UNKNOWN (UNK), which can lead to problems:
  – Joe's wage = salary + commission
    > commission UNK => wage UNK
  – 4 Customers - balance $100 $200 $300 NULL
    > NULL ignored for inbuilt SQL SUM and AVG functions (treated as UNK)
    > tuple counted by COUNT (mathematically AVG = SUM/ROWS)
    > AVG $200 cf SUM $600 No tuples = 4 ?

• Now have values -
  – TRUE, FALSE, UNKNOWN
  – called THREE-VALUED LOGIC
Data integrity requires the database to be an accurate reflection of the real world

Data should be valid and complete

Previously integrity issues have been handled external to the database in the application code.

Codd (1985) states that integrity constraints specific to a particular RDBMS must be definable in the sublanguage and these constraints stored in the system catalogue.
Integrity Constraints

• Integrity rules should be considered at design time

• Transactions must be monitored for integrity violations and appropriate actions taken

• Rules should be few, without overlap and should not impact performance significantly

• Data integrity is all about Trust in the data in a database and the real world it models.
Integrity Constraints

- Originally Codd defined two integrity constraints in the Relational Model – entity and referential integrity. In RM/V2, Codd defined five types of integrity constraints.
  - Type E - Entity Integrity
  - Type R - Referential Integrity
  - Type D - Domain Integrity
  - Type C - Column Integrity
  - Type U - User Defined Integrity
Integrity Constraints

• **Type E or Entity integrity**
  - no primary key value of a base relation is allowed to be null or have a null component.
  - the rationale for this rule was that entities in the real world are distinguishable, that is, they are identifiable in some way. Primary keys perform a unique identification function and therefore must be definitely and unambiguously identifiable
  - does not include in its definition that primary keys must also be unique. This uniqueness requirement is part of the relational model itself (no duplicate tuples)
  - provided by enforcing NOT NULL on the primary key

```sql
CREATE TABLE emp
  (empno NUMBER NOT NULL,
   ename VARCHAR(10) job VARCHAR(9),
   mgr NUMBER,
   ...........
  PRIMARY KEY (empno));
```
Integrity Constraints

- **Type R or Referential integrity**
  - for each distinct, unmarked (not null) foreign-key value in a relational database, there must exist in the database an equal value of a primary key from the same domain
  - if the foreign key is composite, those components that are themselves foreign keys and unmarked (non null) must exist in the database as components of at least one primary-key value drawn from the same domain
  - referential Integrity constraints, through the use of foreign keys allows the representation of the relationships between tables
Integrity Constraints

• **D-type or domain integrity**
  – consists of those integrity constraints that are shared by all the attributes that draw their values from that domain
  – common constraints are regular data types, ranges of values permitted and whether or not the ordering comparators, > and <, are applicable to these values.

• **C-type or column integrity**
  – is a more narrowly defined range linked to domain integrity
  – is specific to a particular attribute in a relation
  – e.g. the PAID attribute in an invoice relation is constrained to the values: Y, y, N, n
Integrity Constraints

- **U-type or user-defined integrity**
  - permit the DBA to define the “business rules” pertaining to the business
  - for example, the salary of an employee should not exceed the salary of the employee’s supervisor
- **Integrity Violations**
  - Violations of integrity constraints of types D, C and E are never permitted. The DBMS must either return a code if the source of the attempted violation is an application program, or the DBMS must deny the request and send a message if the source is a user at a terminal
  - With each R type and U type integrity constraint there must be an accompanying violation response which defines the actions to be taken by the system in case of attempted violation of the constraint. The action may be expressed in either or both the relational language or a host language
Integrity Constraints

• Integrity Violations
  – violation responses for violation of type U depend upon the rules of the business
    > e.g. A student must be enrolled in 4 subjects per semester. If the student overloads or under loads a warning is sent to the user
  – there are a number of suitable violation responses for violation of type R
    > Date (1981) extended the basic idea of referential integrity by introducing a set of foreign key rules which specified what the DBMS should do if an end user attempted to perform an update or delete that would violate a referential constraint
Referential Integrity Rules

• **Referential Integrity Violations**
  - **Delete rule**
    > applies to deleting tuples from the parent (referenced) relation, that is the relation with the primary key
    > for example, deleting a department from the DEPARTMENT relation with matching foreign keys in the EMPLOYEE relation would cause an integrity violation
  - the delete can be restricted, cascaded, nullified or set to a default value
  - **Delete Restrict**
    > The delete operation is restricted or not allowed if a referenced tuple with foreign keys is the target of the delete. For example, deleting a department from the DEPARTMENT relation would be restricted if there were matching foreign keys in the EMPLOYEE relation.
    > Default in Oracle
Referential Integrity Rules

- **Delete Cascade**
  > The delete of a referenced relation cascades to delete all the matching foreign key tuples. For example, deleting a department from the DEPARTMENT relation would cascade to delete all matching foreign keys tuples in the EMPLOYEE relation.

- **Delete Set Null**
  > When a referenced tuple is deleted, then all the matching foreign key values are set to null. For example, deleting a department from the DEPARTMENT relation would result in all matching foreign key values in the EMPLOYEE relation being set to null.
  > can only apply to foreign keys that can accept nulls.

- **Delete Set Default**
  > When a referenced tuple is deleted, then all the matching foreign key values are set to a default value. For example, deleting a department from the DEPARTMENT relation would result in all matching foreign key values in the EMPLOYEE relation being set to a default value.
Referential Integrity Rules

- **Update rule**
  > applies to updating the Primary Key in the parent (referenced) relation
  > for example, updating the department number attribute in the DEPARTMENT relation with matching foreign keys in the EMPLOYEE relation would cause an integrity violation

- **Update Restrict**
  > the update operation is restricted if a referenced tuple exists with a foreign key value equal to a primary key value in the referenced relation. For example, updating the department number in the DEPARTMENT relation would be restricted if there were matching department numbers in the EMPLOYEE relation
  > Default in Oracle
Referential Integrity Rules

- **Update Cascade**
  - the update of a primary key value cascades to update matching foreign key values in dependent relations
  - for example, updating the dept_no in the DEPARTMENT relation would cascade to update all matching dept_nos in the EMPLOYEE relation.

- **Update Set Null**
  - when a primary key is updated, matching foreign key values in dependent relations are set to null
  - can only apply to foreign keys that can accept nulls
  - for example, updating the dept_no in the DEPARTMENT relation would result in all matching dept_nos in the EMPLOYEE relation being set to null.

- **Update Set Default**
  - when a primary key is updated, matching foreign key values in dependent relations are set to a default value
  - for example, updating the dept_no in the DEPARTMENT relation would result in all matching dept_nos in the EMPLOYEE relation being set to a default value.
Referential Integrity

• the referential integrity rule states that the database must not contain any unmatched, non null foreign key values in the referencing (child) relation for which there does not exist a matching value of the primary key in the relevant referenced (parent) relation

• for every relationship between two relations in the database, not only is it necessary to define how updates and deletions of referenced tuples are to be handled but also what to do when a violation of integrity occurs on insertions of referencing tuples or updates of foreign key attributes
Referential Integrity

• when a tuple is inserted into the referencing relation or a foreign key attribute updated in the referencing relation the foreign key value must:
  – either be equal to null (provided nulls are permitted), or
  – be equal to the primary key attribute in the referenced relation
  – for example, when updating the dept_no attribute in the EMPLOYEE relation the value entered must match a dept_no attribute in the DEPARTMENT relation.

• the most suitable response would be either:
  – a failure to insert the new tuple or update the existing foreign key attribute or
  – a ROLLBACK of the database, depending if the update or insert had already occurred
Key Constraints

• **Superkey** (SK) - subset of attributes of R which uniquely identify a tuple
  > \( t_1[SK] \neq t_2[SK] \)
  - SK(Customer) = (custno, custname, custadd, credlimit)
  - SK(Customer) = (custno, custname, custadd)
  - SK(Customer) = (custno, custname)
  - SK(Customer) = (custno)
  > PRIMARY KEY - minimal superkey
  - SK(Orderline) = (orderno, productno)

• **A relation may have more than one key**
  > EMPLOYEE(emp#, ........, taxfile#)
  - Each key \( \Rightarrow \) candidate key (CK)
    » Chosen key \( \Rightarrow \) primary key (PK)
    » Other keys \( \Rightarrow \) alternate keys (AK)
Summary of Keys

- **Superkey**
  - Any key that uniquely identifies each entity

- **Candidate key**
  - Minimal superkey (without redundancies)

- **Primary key**
  - Candidate key (chosen) to uniquely identify all other attributes in a given row

- **Secondary key**
  - Used only for data retrieval

- **Composite key**
  - Composed of more than one attribute

- **Key attribute**
  - Any attribute that is part of a key

- **Foreign key**
  - Values must match a primary key in a referenced (parent) table
Relational Model Operators

• Codd originally defined that access to relational databases would be defined through relational algebra and equivalent relational calculus
• Relational Algebra
  – a procedural query language based on algebraic concepts which define relational operations
  – provides a collection of explicit operations – join, union, project, etc – that can be used to tell the system how to build some desired relation from the given relations in the database
• Relational Calculus
  – non procedural tuple relational calculus
  – provides a notation for formulating the definition of that desired relation in terms of those given relations
  – used by SQL
Example – Get supplier numbers and cities for suppliers who supply part P2

- Relational Algebra:
  > Form the natural join of relations S and SP on S#
  > Next, restrict the result of that join to tuples for part P2
  > Finally, project the result of that restriction on S# and CITY

- Relational Calculus
  > Get S# and CITY for suppliers such that there exists a SHIPMENT SP with the same S# value and with a P# value P2

The calculus formulation is descriptive (states what the problem is) whereas the algebraic one is prescriptive (gives a procedure for solving the problem)
Relational Algebra

- Consists of a collection of high-level operators that operate on relations
- Each operator takes either one or two relations as its input and produces a new relation as output
- Divided into two groups:
  - native operations - focus on structure of relation (heading)
    - select,
    - project,
    - join,
    - division
  - set operations - focus on relations as sets of tuples
    - union,
    - intersection,
    - difference,
    - product
SELECT

- Extracts specified tuples from a specified relation (restricts the specified relation to just those tuples that satisfy a specified condition)
- horizontal subset of a relation
- RESTRICT was the original name for this operation, now too easily confused with the SQL SELECT
- Symbolically:
  - RESULT \( \leftrightarrow \sigma_{\text{predicate}} \text{(relation-name)} \)
- Syntax:
  - SELECT relation-name WHERE condition GIVING result-relation
  - IRA \(<\text{resultname}> = \text{SELECT } <\text{relname}> [ <\text{select_expression}> ]\)
SELECT

• eg. list information from CUSTOMER for customers of salesrep six
  – Symbolic:
    > RESULT \( \leftrightarrow \sigma_{\text{slnumb} = 6} \) (CUSTOMER)
  – Generalised:
    > SELECT CUSTOMER WHERE SLSRNUMB = 6 GIVING RESULT
  – IRA:
    > A1 = SELECT CUSTOMER \{ SLSRNUMB = 6\}

• **SELECT is unary - applied on single relation**
  – Resultant relation:
    > same degree,
    > cardinality \(< \) or = original relation
### SELECT

**Figure 3.9**

**Original table**

<table>
<thead>
<tr>
<th>P_CODE</th>
<th>P_DESCRPT</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456</td>
<td>Flashlight</td>
<td>$5.26</td>
</tr>
<tr>
<td>123457</td>
<td>Lamp</td>
<td>$25.15</td>
</tr>
<tr>
<td>123458</td>
<td>Box Fan</td>
<td>$10.99</td>
</tr>
<tr>
<td>213345</td>
<td>9v battery</td>
<td>$1.92</td>
</tr>
<tr>
<td>254467</td>
<td>100W bulb</td>
<td>$1.47</td>
</tr>
<tr>
<td>311452</td>
<td>Powerdrill</td>
<td>$34.99</td>
</tr>
</tbody>
</table>

**New table or list**

<table>
<thead>
<tr>
<th>P_CODE</th>
<th>P_DESCRPT</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456</td>
<td>Flashlight</td>
<td>$5.26</td>
</tr>
<tr>
<td>123457</td>
<td>Lamp</td>
<td>$25.15</td>
</tr>
<tr>
<td>123458</td>
<td>Box Fan</td>
<td>$10.99</td>
</tr>
<tr>
<td>213345</td>
<td>9v battery</td>
<td>$1.92</td>
</tr>
<tr>
<td>254467</td>
<td>100W bulb</td>
<td>$1.47</td>
</tr>
<tr>
<td>311452</td>
<td>Powerdrill</td>
<td>$34.99</td>
</tr>
</tbody>
</table>

**SELECT ALL yields**

**SELECT only PRICE less than $2.00 yields**

**SELECT only P_CODE = 311452 yields**
PROJECT

• Extracts specified attributes from a specified relation
  – vertical subset of a relation
• Symbolically:
  – RESULT $\pi_{col1, col2, ...}$ (relation-name)
• Syntax:
  – PROJECT relation-name OVER (col1, col2, ...) GIVING result-relation
  – IRA <resultname> = PROJECT <relname> [ <attribute> ... ]
• eg. list customer number and name for all customers
  – Symbolic:
    > RESULT $\pi_{custnumb, custname}$ (CUSTOMER)
  – Generalised
    > PROJECT CUSTOMER OVER (CUSTNUMB, CUSTNAME) GIVING RESULT
  – IRA
    > A2 = PROJECT CUSTOMER [CUSTNUMB, CUSTNAME]
PROJECT

- Attributes of result can be renamed
- Syntax:
  - IRA - create RESULT then rename:
    > REDEFINE <relname> [ <attribute> ... ]
- PROJECT is unary
  - Resultant relation:
    > same cardinality,
    > degree < (or =) original relation
- To list the customer number and name for all the customers of salesrep six ⇒ TWO STAGE's
  - RESULT ✎ \( \pi_{\text{custnumber}, \text{custname}} (\sigma_{\text{slrnumber} = 6} (\text{CUSTOMER})) \)
  - STAGE 1:
    > SELECT CUSTOMER WHERE SLSRNUMB = 6 GIVING HOLD
  - STAGE 2:
    > PROJECT HOLD OVER (CUSTNUMBER, CUSTNAME) GIVING RESULT
### PROJECT

**Figure 3.10**

**Original table**

<table>
<thead>
<tr>
<th>P_CODE</th>
<th>P_DESCRIP</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456</td>
<td>Flashlight</td>
<td>$5.26</td>
</tr>
<tr>
<td>123457</td>
<td>Lamp</td>
<td>$25.15</td>
</tr>
<tr>
<td>123458</td>
<td>Box Fan</td>
<td>$10.99</td>
</tr>
<tr>
<td>213345</td>
<td>9V battery</td>
<td>$1.92</td>
</tr>
<tr>
<td>254467</td>
<td>100W bulb</td>
<td>$1.47</td>
</tr>
<tr>
<td>311452</td>
<td>Powerdrill</td>
<td>$34.99</td>
</tr>
</tbody>
</table>

**New table or list**

**PROJECT PRICE yields**

<table>
<thead>
<tr>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>$5.26</td>
</tr>
<tr>
<td>$25.15</td>
</tr>
<tr>
<td>$10.99</td>
</tr>
<tr>
<td>$1.92</td>
</tr>
<tr>
<td>$1.47</td>
</tr>
<tr>
<td>$34.99</td>
</tr>
</tbody>
</table>

**PROJECT P_DESCRIP and PRICE yields**

**PROJECT P_CODE and PRICE yields**

<table>
<thead>
<tr>
<th>P_CODE</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456</td>
<td>$5.26</td>
</tr>
<tr>
<td>123457</td>
<td>$25.15</td>
</tr>
<tr>
<td>123458</td>
<td>$10.99</td>
</tr>
<tr>
<td>213345</td>
<td>$1.92</td>
</tr>
<tr>
<td>254467</td>
<td>$1.47</td>
</tr>
<tr>
<td>311452</td>
<td>$34.99</td>
</tr>
</tbody>
</table>
Set Operations

• Binary operations
  – two relations as operand's must be union compatible
    > same degree, and
    > attributes in same position in each MUST have same domain
    > set of tuples appearing in both relations

• UNION - \( A \cup B \)
  – Builds a relation consisting of all tuples appearing in either or both of two specified relations
• IRA <resultname> = <relname1> UNION <relname2>
• eg. list all customers who either have orders or are represented by sales rep 12 or both
  – PROJECT ORDERS OVER (CUSTNUMB) GIVING TEMP1
  – SELECT CUSTOMER WHERE SLSRNUMB = 12 GIVING TEMP2
  – PROJECT TEMP2 OVER (CUSTNUMB) GIVING TEMP3
  – UNION TEMP1 WITH TEMP3 GIVING RESULT
### UNION

**Figure 3.5: UNION**

<table>
<thead>
<tr>
<th>P_CODE</th>
<th>P_DESCRPT</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456</td>
<td>Flashlight</td>
<td>$5.26</td>
</tr>
<tr>
<td>123457</td>
<td>Lamp</td>
<td>$25.15</td>
</tr>
<tr>
<td>123458</td>
<td>Box Fan</td>
<td>$10.99</td>
</tr>
<tr>
<td>213345</td>
<td>9V battery</td>
<td>$1.92</td>
</tr>
<tr>
<td>254467</td>
<td>100W bulb</td>
<td>$1.47</td>
</tr>
<tr>
<td>311452</td>
<td>Powerdrill</td>
<td>$34.99</td>
</tr>
</tbody>
</table>

**UNION**

<table>
<thead>
<tr>
<th>P_CODE</th>
<th>P_DESCRPT</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>345678</td>
<td>Microwave</td>
<td>$160.00</td>
</tr>
<tr>
<td>345679</td>
<td>Dishwasher</td>
<td>$500.00</td>
</tr>
</tbody>
</table>

Yields:

<table>
<thead>
<tr>
<th>P_CODE</th>
<th>P_DESCRPT</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456</td>
<td>Flashlight</td>
<td>$5.26</td>
</tr>
<tr>
<td>123457</td>
<td>Lamp</td>
<td>$25.15</td>
</tr>
<tr>
<td>123458</td>
<td>Box Fan</td>
<td>$10.99</td>
</tr>
<tr>
<td>213345</td>
<td>9V battery</td>
<td>$1.92</td>
</tr>
<tr>
<td>254467</td>
<td>100W bulb</td>
<td>$1.47</td>
</tr>
<tr>
<td>311452</td>
<td>Powerdrill</td>
<td>$34.99</td>
</tr>
<tr>
<td>345678</td>
<td>Microwave</td>
<td>$160.00</td>
</tr>
<tr>
<td>345679</td>
<td>Dishwasher</td>
<td>$500.00</td>
</tr>
</tbody>
</table>

---

44
**PRODUCT - Cartesian Product**

- Builds a relation from two specified relations consisting of all possible combinations of tuples, one from each of the two relations
- Concatenate every row of the first table with every row of the second table

**A**

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
</tr>
<tr>
<td>a2</td>
<td>b2</td>
<td>c2</td>
</tr>
<tr>
<td>a3</td>
<td>b3</td>
<td>c3</td>
</tr>
</tbody>
</table>

**B**

<table>
<thead>
<tr>
<th>a</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>d1</td>
<td>e1</td>
</tr>
<tr>
<td>a2</td>
<td>d2</td>
<td>e2</td>
</tr>
</tbody>
</table>

**A X B**

<table>
<thead>
<tr>
<th>A.a</th>
<th>b</th>
<th>c</th>
<th>B.a</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>a1</td>
<td>d1</td>
<td>e1</td>
</tr>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>a2</td>
<td>d2</td>
<td>e2</td>
</tr>
<tr>
<td>a2</td>
<td>b2</td>
<td>c2</td>
<td>a1</td>
<td>d1</td>
<td>e1</td>
</tr>
<tr>
<td>a2</td>
<td>b2</td>
<td>c2</td>
<td>a2</td>
<td>d2</td>
<td>e2</td>
</tr>
<tr>
<td>a3</td>
<td>b3</td>
<td>c3</td>
<td>a1</td>
<td>d1</td>
<td>e1</td>
</tr>
<tr>
<td>a3</td>
<td>b3</td>
<td>c3</td>
<td>a2</td>
<td>d2</td>
<td>e2</td>
</tr>
</tbody>
</table>
**PRODUCT**

Yields all possible pairs from two tables

**FIGURE 3.8**

<table>
<thead>
<tr>
<th>P_CODE</th>
<th>P_DESCRPT</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>20456</td>
<td>Flashlight</td>
<td>$5.26</td>
</tr>
<tr>
<td>123457</td>
<td>Lamp</td>
<td>$25.15</td>
</tr>
<tr>
<td>123458</td>
<td>Box Fan</td>
<td>$10.99</td>
</tr>
<tr>
<td>213345</td>
<td>9v battery</td>
<td>$1.92</td>
</tr>
<tr>
<td>254467</td>
<td>100V bulb</td>
<td>$1.47</td>
</tr>
<tr>
<td>311452</td>
<td>Powerdrill</td>
<td>$34.99</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STORE</th>
<th>AISLE</th>
<th>SHELF</th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td>W</td>
<td>5</td>
</tr>
<tr>
<td>m</td>
<td>M</td>
<td>5</td>
</tr>
<tr>
<td>24</td>
<td>K</td>
<td>9</td>
</tr>
<tr>
<td>25</td>
<td>Z</td>
<td>6</td>
</tr>
</tbody>
</table>

The figure shows two tables of products with their codes, descriptions, and prices. The figure illustrates how these tables can be paired to yield all possible combinations. The arrows indicate the process of combining the two tables.
JOIN

• Builds a relation from two specified relations consisting of all possible combinations of tuples, one from each of the two relations, such that the two tuples contributing to any given combination satisfy some specified condition
• Syntax:
  – relation-name1 TIMES relation-name2 GIVING result-relation
  – IRA <resultname> = <relname1> CROSS <relname2>

• CARTESIAN PRODUCT is the basis for the JOIN operation (a further native operation)

• The join operation comes in several different varieties
JOIN

• **THETA-JOIN** or **GENERAL JOIN**
  – join two relations together on the basis of some condition other than equality
  – Symbolically:
    > \( \text{RESULT } \bowtie R \bowtie S \)
    > The predicate (join condition) is of the form \( R.ai \ q S.bi \) where \( q \) may be one of the comparison operators \( (<, \leq, >, \geq, =, \sim=) \)
  – This is equivalent to
    > \( \sigma<\text{join condition}>(R \times S) \)
  – Syntax:
    > IRA <resultname> = JOIN <relname1> [ <attribute1> <op> <attribute2> ] <relname2>

• **Special cases of theta join where condition is equal** is known as an **EQUI-JOIN**
EQUI and NATURAL JOIN

- **EQUIJOIN** - join on common attribute, join column appears twice
  - Symbolically:
    > RESULT $\leftrightarrow$ A $\bowtie$ a.column = b.column B
  - Note that this is equivalent to
    > $\sigma$ a.column = b.column (A X B)
  - Syntax:
    > relation-name1 EQUIJOIN relation-name2 GIVING result-relation

- **NATURAL JOIN** - join on common domain (normally common attribute), join column only appears once
  - Syntax:
    > relation-name1 JOIN relation-name2 GIVING result-relation
    > IRA <resultname> = JOIN <relname1> [ <attribute> ] <relname2>
**FIGURE 3.11**

Two tables that will be used in join illustrations

<table>
<thead>
<tr>
<th>CUS_CODE</th>
<th>CUS_NAME</th>
<th>CUS_ZIP</th>
<th>AGENT_CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1132445</td>
<td>Walker</td>
<td>32145</td>
<td>231</td>
</tr>
<tr>
<td>1217782</td>
<td>Andaes</td>
<td>32145</td>
<td>125</td>
</tr>
<tr>
<td>1512243</td>
<td>Rakowski</td>
<td>34129</td>
<td>167</td>
</tr>
<tr>
<td>1321247</td>
<td>Rodriguez</td>
<td>37134</td>
<td>125</td>
</tr>
<tr>
<td>1542311</td>
<td>Smithson</td>
<td>37134</td>
<td>421</td>
</tr>
<tr>
<td>1657399</td>
<td>Vanloon</td>
<td>32145</td>
<td>231</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AGENT_CODE</th>
<th>AGENT_PHONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>6152435887</td>
</tr>
<tr>
<td>167</td>
<td>6153426778</td>
</tr>
<tr>
<td>231</td>
<td>6152431124</td>
</tr>
<tr>
<td>333</td>
<td>9041234445</td>
</tr>
</tbody>
</table>
Natural Join

- Links tables by selecting only rows with common values in their common attribute(s)
- Result of a three-stage process:
  1. PRODUCT of the tables is created
  2. SELECT is performed on Step 1 output to yield only the rows for which the AGENT_CODE values are equal
     > Common column(s) are called join column(s)
  3. PROJECT is performed on Step 2 results to yield a single copy of each attribute, thereby eliminating duplicate columns
Natural Join

<table>
<thead>
<tr>
<th>CUS_CODE</th>
<th>CUS_NAME</th>
<th>CUS_ZIP</th>
<th>CUSTOMER_AGENT_CODE</th>
<th>AGENT_AGENT_CODE</th>
<th>AGENT_PHONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>132445</td>
<td>Walker</td>
<td>32145</td>
<td>231</td>
<td>125</td>
<td>6152439887</td>
</tr>
<tr>
<td>132445</td>
<td>Walker</td>
<td>32145</td>
<td>231</td>
<td>167</td>
<td>6153426778</td>
</tr>
<tr>
<td>132445</td>
<td>Walker</td>
<td>32145</td>
<td>231</td>
<td>231</td>
<td>6152431124</td>
</tr>
<tr>
<td>132445</td>
<td>Walker</td>
<td>32145</td>
<td>231</td>
<td>333</td>
<td>9041234445</td>
</tr>
<tr>
<td>1217782</td>
<td>Adaes</td>
<td>32145</td>
<td>125</td>
<td>125</td>
<td>6152439887</td>
</tr>
<tr>
<td>1217782</td>
<td>Adaes</td>
<td>32145</td>
<td>125</td>
<td>167</td>
<td>6153426778</td>
</tr>
<tr>
<td>1217782</td>
<td>Adaes</td>
<td>32145</td>
<td>125</td>
<td>231</td>
<td>6152431124</td>
</tr>
<tr>
<td>1217782</td>
<td>Adaes</td>
<td>32145</td>
<td>125</td>
<td>333</td>
<td>9041234445</td>
</tr>
<tr>
<td>1312243</td>
<td>Rakowski</td>
<td>34129</td>
<td>167</td>
<td>125</td>
<td>6152439887</td>
</tr>
<tr>
<td>1312243</td>
<td>Rakowski</td>
<td>34129</td>
<td>167</td>
<td>167</td>
<td>6153426778</td>
</tr>
<tr>
<td>1312243</td>
<td>Rakowski</td>
<td>34129</td>
<td>167</td>
<td>231</td>
<td>6152431124</td>
</tr>
<tr>
<td>1312243</td>
<td>Rakowski</td>
<td>34129</td>
<td>167</td>
<td>333</td>
<td>9041234445</td>
</tr>
<tr>
<td>1321242</td>
<td>Rodriguez</td>
<td>37134</td>
<td>125</td>
<td>125</td>
<td>6152439887</td>
</tr>
<tr>
<td>1321242</td>
<td>Rodriguez</td>
<td>37134</td>
<td>125</td>
<td>167</td>
<td>6153426778</td>
</tr>
<tr>
<td>1321242</td>
<td>Rodriguez</td>
<td>37134</td>
<td>125</td>
<td>231</td>
<td>6152431124</td>
</tr>
<tr>
<td>1321242</td>
<td>Rodriguez</td>
<td>37134</td>
<td>125</td>
<td>333</td>
<td>9041234445</td>
</tr>
<tr>
<td>1542311</td>
<td>Smithson</td>
<td>37134</td>
<td>421</td>
<td>125</td>
<td>6152439887</td>
</tr>
<tr>
<td>1542311</td>
<td>Smithson</td>
<td>37134</td>
<td>421</td>
<td>167</td>
<td>6153426778</td>
</tr>
<tr>
<td>1542311</td>
<td>Smithson</td>
<td>37134</td>
<td>421</td>
<td>231</td>
<td>6152431124</td>
</tr>
<tr>
<td>1542311</td>
<td>Smithson</td>
<td>37134</td>
<td>421</td>
<td>333</td>
<td>9041234445</td>
</tr>
<tr>
<td>1567399</td>
<td>Vanioc</td>
<td>32145</td>
<td>231</td>
<td>125</td>
<td>6152439887</td>
</tr>
<tr>
<td>1567399</td>
<td>Vanioc</td>
<td>32145</td>
<td>231</td>
<td>167</td>
<td>6153426778</td>
</tr>
<tr>
<td>1567399</td>
<td>Vanioc</td>
<td>32145</td>
<td>231</td>
<td>231</td>
<td>6152431124</td>
</tr>
<tr>
<td>1567399</td>
<td>Vanioc</td>
<td>32145</td>
<td>231</td>
<td>333</td>
<td>9041234445</td>
</tr>
</tbody>
</table>
Natural Join

**FIGURE 3.13** Natural join, Step 2: SELECT

<table>
<thead>
<tr>
<th>CUS_CODE</th>
<th>CUS_LNAME</th>
<th>CUS_ZIP</th>
<th>CUSTOMER_AGENT_CODE</th>
<th>AGENT_AGENT_CODE</th>
<th>AGENT_PHONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>217782</td>
<td>Adares</td>
<td>32145</td>
<td>125</td>
<td>125</td>
<td>6152439887</td>
</tr>
<tr>
<td>1321242</td>
<td>Rodriguez</td>
<td>37134</td>
<td>125</td>
<td>125</td>
<td>6152439887</td>
</tr>
<tr>
<td>1312243</td>
<td>Rakowski</td>
<td>34129</td>
<td>167</td>
<td>167</td>
<td>6153426778</td>
</tr>
<tr>
<td>1132445</td>
<td>Walker</td>
<td>32145</td>
<td>231</td>
<td>231</td>
<td>6152431124</td>
</tr>
<tr>
<td>1657399</td>
<td>Vanloo</td>
<td>32145</td>
<td>231</td>
<td>231</td>
<td>6152431124</td>
</tr>
</tbody>
</table>

**FIGURE 3.14** Natural join, Step 3: PROJECT

<table>
<thead>
<tr>
<th>CUS_CODE</th>
<th>CUS_LNAME</th>
<th>CUS_ZIP</th>
<th>AGENT_CODE</th>
<th>AGENT_PHONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>217782</td>
<td>Adares</td>
<td>32145</td>
<td>125</td>
<td>6152439887</td>
</tr>
<tr>
<td>1321242</td>
<td>Rodriguez</td>
<td>37134</td>
<td>125</td>
<td>6152439887</td>
</tr>
<tr>
<td>1312243</td>
<td>Rakowski</td>
<td>34129</td>
<td>167</td>
<td>6153426778</td>
</tr>
<tr>
<td>1132445</td>
<td>Walker</td>
<td>32145</td>
<td>231</td>
<td>6152431124</td>
</tr>
<tr>
<td>1657399</td>
<td>Vanloo</td>
<td>32145</td>
<td>231</td>
<td>6152431124</td>
</tr>
</tbody>
</table>
Natural Join

• **Final outcome yields table that**
  – Does not include unmatched pairs
  – Provides only copies of matches

• **If no match is made between the table rows**
  – the new table does not include the unmatched row

• **The column on which we made the JOIN—that is, AGENT_CODE—occurs only once in the new table**

• **If the same AGENT_CODE were to occur several times in the AGENT table**
  – a customer would be listed for each match
More complicated queries require repeated use of SELECT, PROJECT and JOIN

eg. Find the products listed on each order placed by customer 123, show for each order number the associated product numbers

Symbolically

\[ \pi \text{ordnumb, prodnumb} (\sigma \text{custnumb} = 123 (\text{ORDERS} \mid \times \mid \text{ORDLNE})) \]

OR

\[ \pi \text{ordnumb, prodnumb} ((\sigma \text{custnumb} = 123 (\text{ORDERS})) \mid \times \mid \text{ORDLNE}) \]

IRA:

\[ a1 = \text{select orders}\ [\text{custnumb} = 123] \]
\[ a2 = \text{join } a1 \ [\text{ordnumb}] \text{ ordlne} \]
\[ a3 = \text{project } a2 \ [\text{ordnumb, prodnumb}] \]
Outer Join

- Matched pairs are retained and any unmatched values in other table are left null

- In outer join for tables CUSTOMER and AGENT, two scenarios are possible:
  - Left outer join
    > Yields all rows in CUSTOMER table, including those that do not have a matching value in the AGENT table
  - Right outer join
    > Yields all rows in AGENT table, including those that do not have matching values in the CUSTOMER table

- The label left and right refer to the order in which the tables are listed in the SQL command
Outer Join

**FIGURE 3.15**  
Left outer join

<table>
<thead>
<tr>
<th>CUS_CODE</th>
<th>CUS_LNAME</th>
<th>CUS_ZIP</th>
<th>AGENT_CODE</th>
<th>AGENT_PHONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1217782</td>
<td>Amapes</td>
<td>32145</td>
<td>125</td>
<td>6152439887</td>
</tr>
<tr>
<td>1321242</td>
<td>Rodriguez</td>
<td>37134</td>
<td>125</td>
<td>6152439887</td>
</tr>
<tr>
<td>1312243</td>
<td>Rakowski</td>
<td>34129</td>
<td>167</td>
<td>6153426778</td>
</tr>
<tr>
<td>1132445</td>
<td>Walker</td>
<td>32145</td>
<td>231</td>
<td>6152431124</td>
</tr>
<tr>
<td>1657399</td>
<td>Vanlloo</td>
<td>32145</td>
<td>231</td>
<td>6152431124</td>
</tr>
<tr>
<td>1542311</td>
<td>Smithson</td>
<td>37134</td>
<td>421</td>
<td></td>
</tr>
</tbody>
</table>

**FIGURE 3.16**  
Right outer join

<table>
<thead>
<tr>
<th>CUS_CODE</th>
<th>CUS_LNAME</th>
<th>CUS_ZIP</th>
<th>AGENT_CODE</th>
<th>AGENT_PHONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1217782</td>
<td>Amapes</td>
<td>32145</td>
<td>125</td>
<td>6152439887</td>
</tr>
<tr>
<td>1321242</td>
<td>Rodriguez</td>
<td>37134</td>
<td>125</td>
<td>6152439887</td>
</tr>
<tr>
<td>1312243</td>
<td>Rakowski</td>
<td>34129</td>
<td>167</td>
<td>6153426778</td>
</tr>
<tr>
<td>1132445</td>
<td>Walker</td>
<td>32145</td>
<td>231</td>
<td>6152431124</td>
</tr>
<tr>
<td>1657399</td>
<td>Vanlloo</td>
<td>32145</td>
<td>231</td>
<td>6152431124</td>
</tr>
<tr>
<td>1542311</td>
<td>Smithson</td>
<td>37134</td>
<td>333</td>
<td>9041234445</td>
</tr>
</tbody>
</table>
Query Optimisation

- **Tuple relational calculus (SQL)**
  - converted into relational algebra operations by DBMS
  - most ‘efficient’ solution path selected
- For each order placed by customer 123 list the order number, the product description of each product ordered and the quantity ordered

```
SELECT ordnumb, prod_desc, qty
FROM orders o, ordline l, product p
WHERE o.ordnumb = l.ordnumb
  AND l.prodnumb = p.prodnumb
  AND o.custnumb = 1234
```

SQL – notice, non procedural what is wanted, not how to get
Query Optimisation – heuristic solution

Canonical query

‘Improved’ query
Data Dictionary and System Catalog

- **Data dictionary**
  - Provides detailed account of all tables found within database
  - Metadata
  - Attribute names and characteristics

- **System catalog**
  - Detailed data dictionary
  - System-created database
  - Stores database characteristics and contents
  - Tables can be queried just like any other tables
  - Automatically produces database documentation
• **Oracle**

```sql
SQL> SELECT tname
    2  FROM sys.syscatalog
    3  WHERE tabletype='TABLE'
    4  AND (creator='SYS'
    5  OR creator='SYSTEM');

+----------+
<table>
<thead>
<tr>
<th>TNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICOL$</td>
</tr>
<tr>
<td>CON$</td>
</tr>
<tr>
<td>UNDO$</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>1191 rows selected.</td>
</tr>
</tbody>
</table>
```
A Sample Data Dictionary

<table>
<thead>
<tr>
<th>TABLE NAME</th>
<th>ATTRIBUTE NAME</th>
<th>CONTENTS</th>
<th>TYPE</th>
<th>FORMAT</th>
<th>RANGE</th>
<th>REQUIRED</th>
<th>PK OR FK</th>
<th>FK REFERENCED TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUSTOMER</td>
<td>CUS_CODE</td>
<td>Customer account code</td>
<td>CHAR(5)</td>
<td>99999</td>
<td>10000–99999</td>
<td>Y</td>
<td>PK</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CUS_LNAME</td>
<td>Customer last name</td>
<td>VCHAR(20)</td>
<td>X</td>
<td>X</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CUS_FNAME</td>
<td>Customer first name</td>
<td>VCHAR(20)</td>
<td>X</td>
<td>X</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CUS_INITIAL</td>
<td>Customer initial</td>
<td>CHAR(1)</td>
<td>X</td>
<td>X</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CUS_RENEW_DATE</td>
<td>Customer insurance renewal date</td>
<td>DATE</td>
<td>X</td>
<td>X</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AGENT_CODE</td>
<td>Agent code</td>
<td>CHAR(3)</td>
<td>999</td>
<td>100–999</td>
<td>Y</td>
<td>Y</td>
<td>FK</td>
</tr>
<tr>
<td></td>
<td>AGENT_AREA_CODE</td>
<td>Agent area code</td>
<td>CHAR(3)</td>
<td>999</td>
<td>100–999</td>
<td>Y</td>
<td>Y</td>
<td>FK</td>
</tr>
<tr>
<td></td>
<td>AGENT_PHONE</td>
<td>Agent telephone number</td>
<td>CHAR(8)</td>
<td>999-9999</td>
<td>0.00 – 9,999,999,999</td>
<td>Y</td>
<td>Y</td>
<td>FK</td>
</tr>
<tr>
<td></td>
<td>AGENT_LNAME</td>
<td>Agent last name</td>
<td>VCHAR(20)</td>
<td>X</td>
<td>X</td>
<td>Y</td>
<td>Y</td>
<td>FK</td>
</tr>
<tr>
<td></td>
<td>AGENT_YTD_SLS</td>
<td>Agent year-to-date sales</td>
<td>NUMBER(9,2)</td>
<td>9,999,999,999</td>
<td>9,999,999,999,999</td>
<td>Y</td>
<td>Y</td>
<td>FK</td>
</tr>
</tbody>
</table>

**Note:** Telephone area codes are always composed of digits 0–9. Because area codes are not used arithmetically, they are most efficiently stored as character data. Also, the area codes are always composed of three digits. Therefore, the area code data type is defined as CHAR(3). On the other hand, names do not conform to some standard length. Therefore, the customer first names are defined as VARCHAR(20), thus indicating that up to 20 characters may be used to store the names. Character data are shown as left-justified.

FK = Foreign key
PK = Primary key
CHAR = Fixed character length data (1–255 characters)
VARCHAR = Variable character length data (1–2,000 characters)
NUMBER = Numeric data (NUMBER(9,2) is used to specify numbers with two decimal places and up to nine digits, including the decimal places. Some RDBMSs permit the use of a MONEY or CURRENCY data type.)
Indexes

- Is an ordered array composed of index key values and row ID values (pointers)
- Generally used to speed up queries
- Primary Key
  - When defined the DBMS automatically creates a unique index on the primary key column(s) you declare
- A table can have many indexes but each index is associated with only one table
- The index key can have multiple attributes
- Index can be created by designer on Secondary Key (SK) to speed access eg. Retrieval via student name
Indexes

Secondary Index
Index on PAINTER_NUM
Summary

• **This lecture**
  – Explain the features of the Relational Model: Components of a relation, Properties of a relation, Null values, and Relational Operators
  – Make use of a subset of the relational operators to solve data queries using symbolic notation: SELECT, PROJECT and JOIN
  – Appreciate that the relational database model takes a logical view of data and how data redundancy is dealt with
  – Describe the basic relational database components: Entities, Attributes, Relationships amongst entities, Integrity constraints, and Data Dictionary
  – Describe the relational table's components and characteristics and contrast the table with a relation
  – Explain how keys are used in the relational database environment: candidate keys, primary keys, alternate keys, foreign keys, and secondary keys
  – Describe the role of an index in the relational database model

• **Next lecture**
  – Conceptual Data Modelling