# پوهنتون کابل

# پوهنځی کمپیوترساینس

دیپارتمنت سیستم های معلوماتی

# Relational Data Manipulating Languages

# Lecture 01 - 03

تهیه کننده : پوهنیار محمد شعیب "زرین خیل"

سال : 1389

# Relational Data Manipulating Languages
# 01

By: M Shuaib Zarinkhail 2010

# Properties of Relation

- Each column has a unique name which shows one attribute of an entity
- All values in a column must be of the same type (Data Type)
- Order of columns and rows is irrelevant
- Each cell in a relation must be single value
  - No repeating  groups or arrays in one cell
- No two rows can be identical

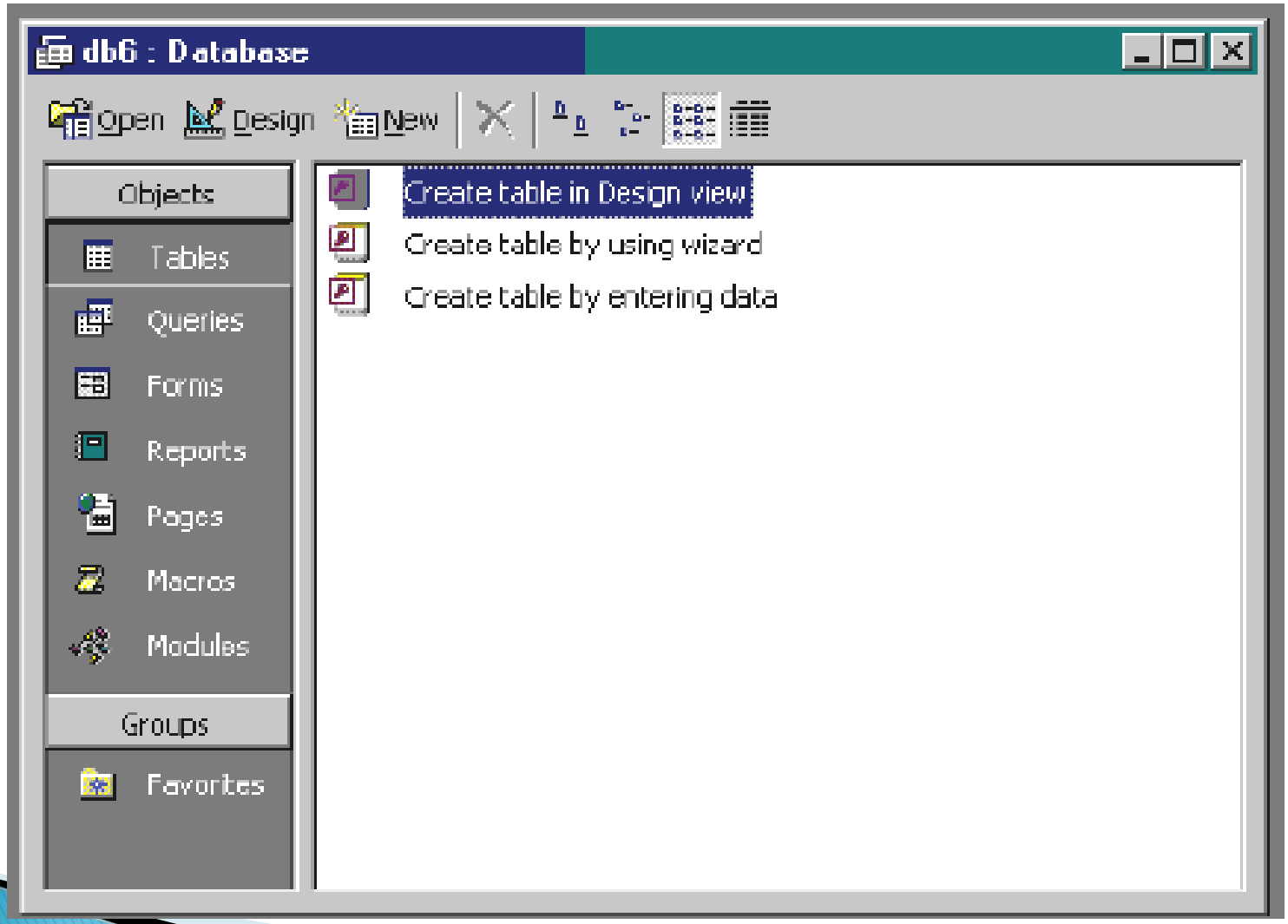
# Relation - Example

| National ID | Name | Date of Birth |
|---|---|---|
| 123-45-4567 | Saber | 12-03-1354 |
| 123-45-5678 | Maryam | 04-11-1358 |
| 123-45-6789 | Kaihan | 06-11-1364 |

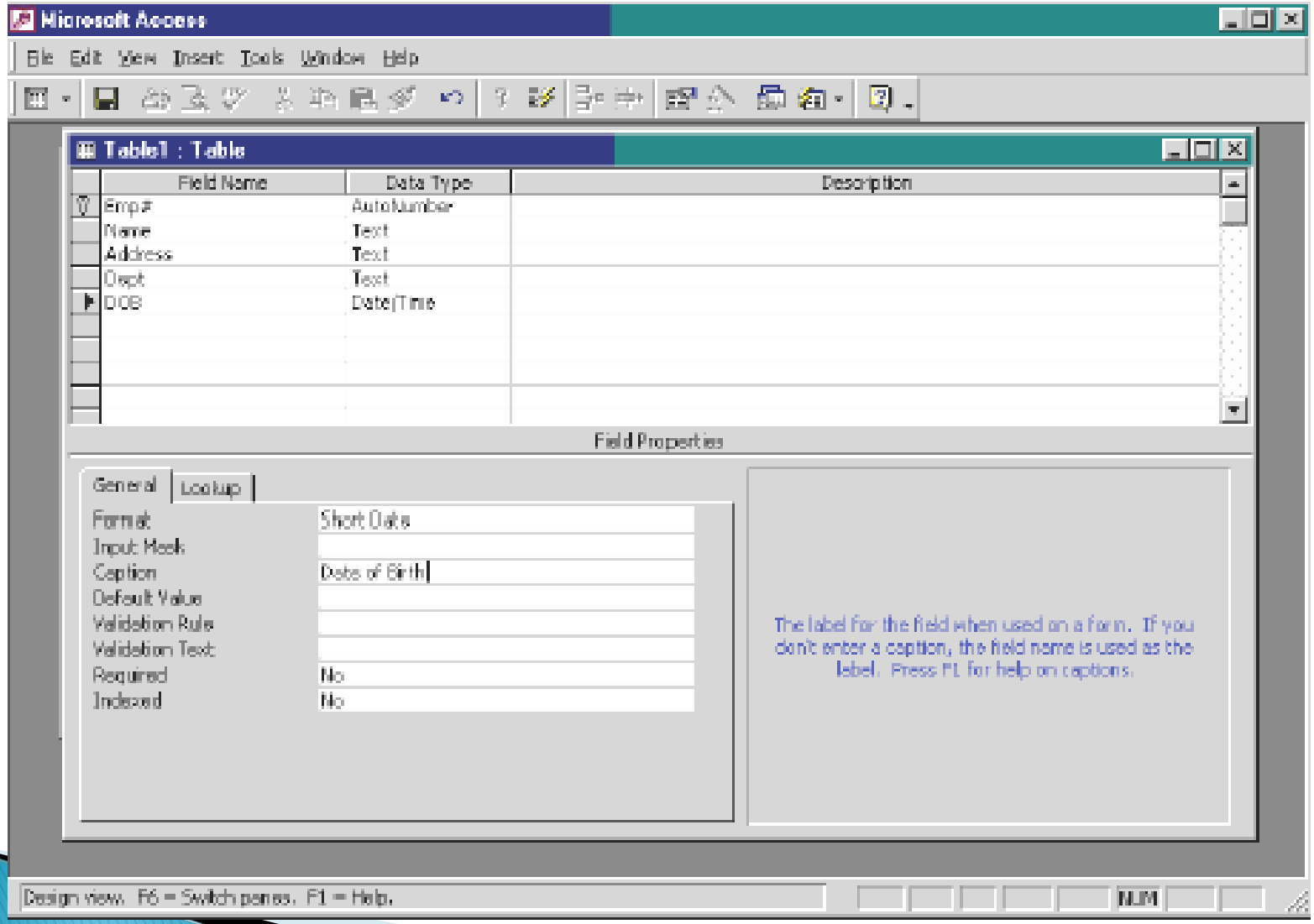
# Database Definition

- Prior to use, a database must be created
  - Space must be allocated for tables, userdata, indexes, and metadata
  - Metadata tables need to be created and populated
- Different ways to create the DB structure:
  - GUI (Graphical User Interface)
  - SQL DDL (SQL – Data Definition Language)

# GUI


db6 : Database
Open
Design
New
Objects
Tables
Queries
Forms
Reports
Pages
Macros
Modules
Groups
Favorites
Create table in Design view
Create table by using wizard
Create table by entering data

# GUI


Microsoft Access
File Edit View Insert Tools Window Help
Table1 : Table
Field Name
Data Type
Description
Emp#
AutoNumber
Name
Text
Address
Text
Dept
Text
DOB
Date/Time
Field Properties
General
Lookup
Format
Short Date
Input Mask
Caption
Date of Birth
Default Value
Validation Rule
Validation Text
Required
No
Indexed
No
The label for the field when used on a form. If you don't enter a caption, the field name is used as the label. Press F1 for help on captions.
Design view. F6 = Switch panes. F1 = Help.
NUM

# SQL DDL

```
CREATE TABLE EMPLOYEE(
        emp#        integer             not null,
        name        varchar(40)         not null,
        address     varchar(80)         not null,
        dept        varchar(20),
        dob         date
  );
```

# Relational Data Manipulation Languages

- Examples of relational data manipulation languages:
  - Relational Algebra
  - Relational Calculus
  - QBE / QBF
  - Transform-Oriented Languages

# Relational Algebra (RA)

- Procedural
- Part of Computer Science
- Part of Relational Model
- Deals with sets of relations

# Relational Algebra (RA)

- Relevant to Mathematical Logic and Set Theory
- Relations can be manipulated using operators
- Relations deal as relvars
  - Topic of much of this and next lectures

# Relational Calculus (RC)

- Non-procedural
- Can be used for proofs
- Mainly used by theorists
- Comes from predicate calculus

Example:

F2 = ($Y)(p(X,Y) Ú q(Y,Z))

# Relational Calculus (RC)

- Consists of two calculi
  - The tuple relational calculus (TRC)
    - Variables range over tuples
  - The domain relational calculus (DRC)
    - Variables range over domain elements (field values)

# Relational Calculus (RC)

- RC has variables, constants, comparison operators, logical connectives and quantifiers
- Expressions in the RC are called formulas
- RC is contrast to RA

# RA Vs RC

- RA and RC are logically equivalent
- For any algebraic expression in **RA**, there is an equivalent expression in **RC** and vice versa
- The difference is known as Dr Codd's Theorem

# RA Vs RC (Example)

- RA might suggest these steps in a query
  - Join books and titles over the BookStoreID
  - Restrict the result of that join to tuples for *a sample book*
  - Project the result of that over StoreName and StorePhone
- RC would formulate a descriptive, declarative way:
  - Get StoreName and StorePhone for supplies such that there exists a title BK with the same BookStoreID value and with a BookTitle value of *a Sample book*

# Transform-Oriented Languages

- Non-procedural languages
- Produce results as a relation
- Mostly extracted from SQL
  - SQL stands for Structured Query Language
- Examples are in the next slide

# Transform Oriented Languages (Examples)

- .QL (Dot Quel)
- CQL (Common Query Language)
- ERROL (Entity Relationship Role Oriented Language)
- QUEL
- QBE (Query By Example)
- SPARQL (Sparkle)
- XQuery (XML Query)
- XSQL (XMLSQL)

# Query-by-Example (QBE)

- Is a database query language for relational DBs
- First developed at IBM in the early 1970's by "Moshe M. Zloof"
- QBE is both the name of the DML and an early database system that included this language
- QBE is the first graphical query language, using visual tables where the user would enter commands, example elements and conditions

# QBE

QBE Has two distinct features:

2. a two dimensional syntax
   - Queries look like tables
3. queries are expressed “by example”, Instead of giving procedures for obtaining a desired answer
   - the user gives an example of what is desired

# QBE

- There are two flavors of QBE

3. GUI based methods of dealing with a DB
   - Graphical version (i.e. Microsoft-Access)

4. By filling in a form or spreadsheet
   - The original text based version
   - Can alter the database or query

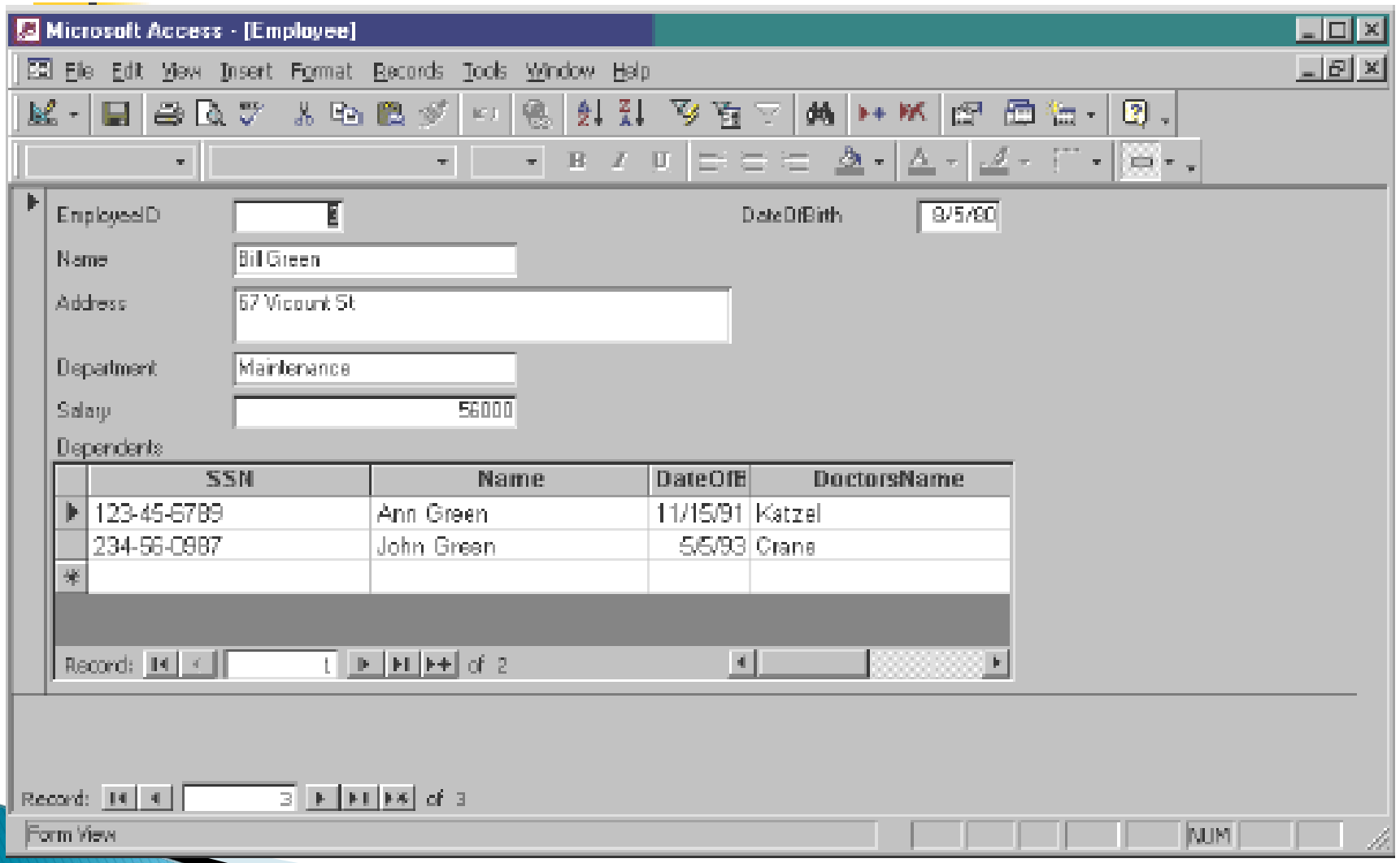
# Manipulating Data in a DBMS

- Major methods for manipulating data:
  - Forms
  - Query languages
  - Stored procedures
  - Application Program Interfaces (API)

# Forms

- Common way to control access in a database
- Usually custom-built for a table or set of tables
- Many databases have this type of feature:
  - Oracle (additional package)
  - MS-Access (standard with the database)

# Sample Form (MS-Access)



# Query languages

- SQL is the most common query language
- SQL is non-procedural
  - Lets database determine how to get the data
- Usually used via a client/server DB
- Can perform other operations aside from queries: update, insert, and delete data

# Sample SQL Statements

- SELECT emp#, name, address
  FROM EMPLOYEE
  WHERE dept = “Accounting” OR dept = “Design”;
  - Queries employee number, name and address for those who belong to the “Accounting” or “Design” departments
- UPDATE EMPLOYEE
  SET salary = salary * 1.05
  WHERE name = “Feda”;
  - This query adds a 5% increase to the salaries of Feda

# Stored Procedures

- Most DBs provide a way to store commonly used SQL statements
- Analogous to the idea of a function or method in a programming language
- Can pass in arguments to fill in parameters within the stored procedure
- Could contain multiple SQL statements

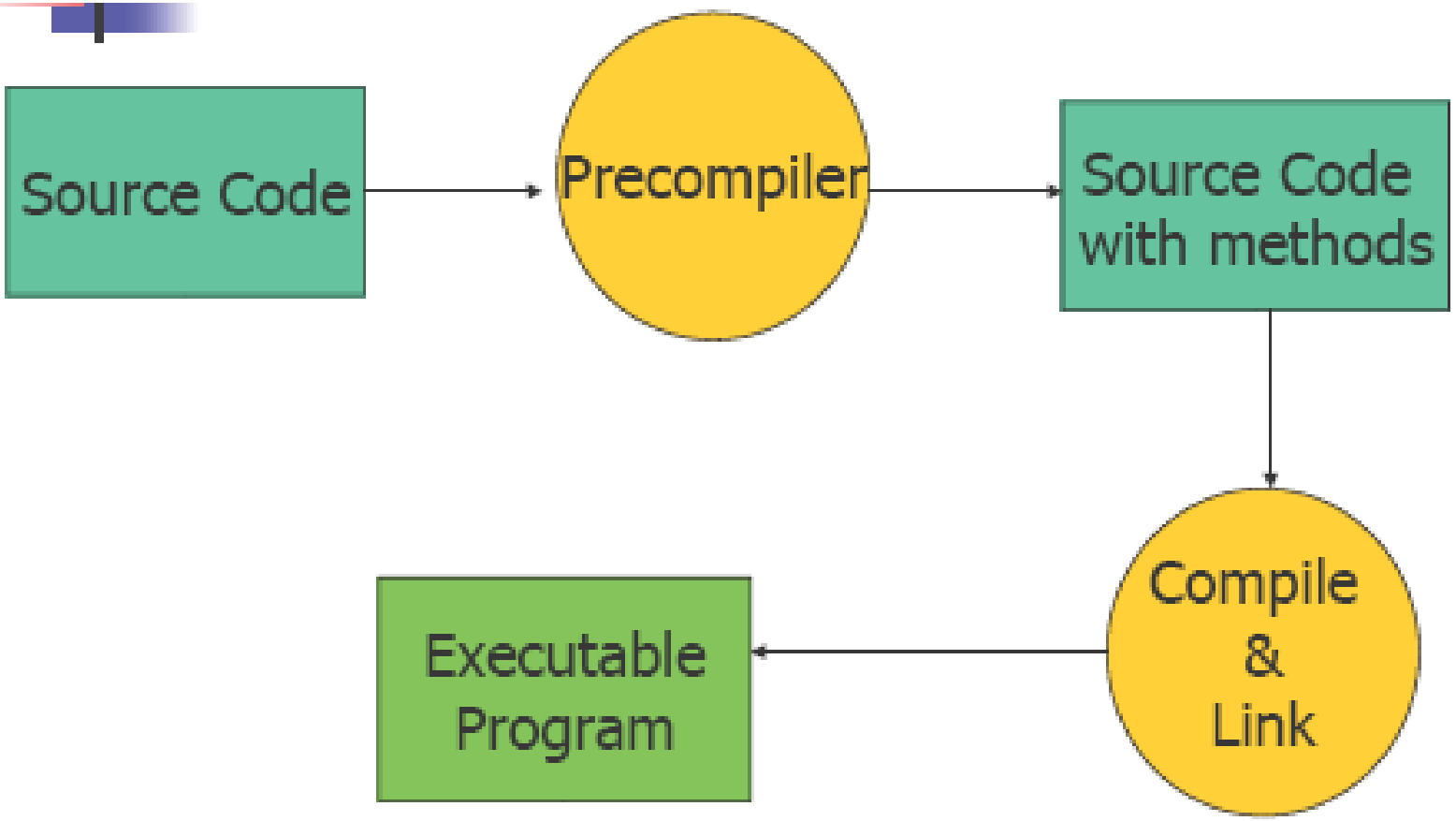
# Application Program Interfaces (API)

- Many DBs have APIs available that allow developers to write programs that will connect up to the DB
  - Example: Java's JDBC
- APIs allow developers to:
  - Issue SQL commands
  - Work with the results that are returned by the DB

# APIs

- Often have an underlying set of methods that are actually used
- Methods and usage documents are included
- A pre-compiler is used to convert the SQL statements in the appropriate method calls

# APIs


Source Code
Precompiler
Source Code with methods
Compile & Link
Executable Program

# Relational Algebra (RA)
# 02

By: M Shuaib Zarinkhail 2010

# Relational Algebra

- Received little attention until the publication of Codd's Relational Model Theory in 1970
- Codd proposed such an algebra as a basis for DB Query Languages
- The first query language based on Codd's algebra was ISBL (Information Systems Base Language)

# Relational Algebra

- Provides a way to construct new relations from existing relations
- Query languages are based on RA
- Deals as "Read-Only" to the relations
- Some operations are similar to the corresponding set operations

# Relational Algebra

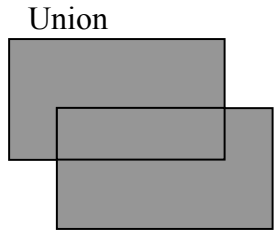
Consists of:

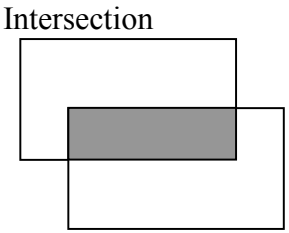
- variables
  - represent relations
- Operators
  - represent the actions we can take with these relations

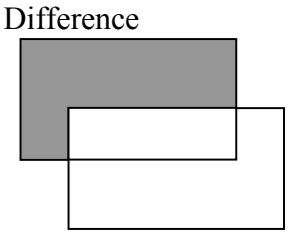
# RA Operators

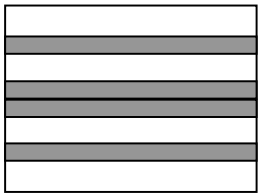
- Three classes of operators:
  - Set based:
    - Union
    - Intersection
    - Difference
  - Operations to remove parts of relations:
    - Selection: eliminates some rows
    - Projection: eliminates some columns
  - Operations that join tuples in two relations:
    - Product
    - Join
    - Divide

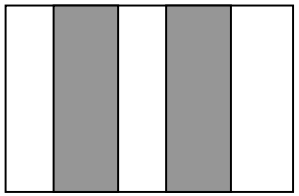
# RA Operators

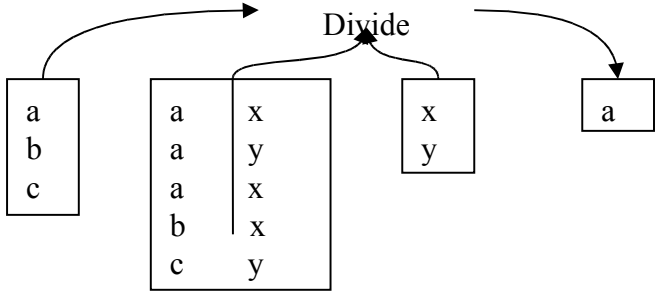


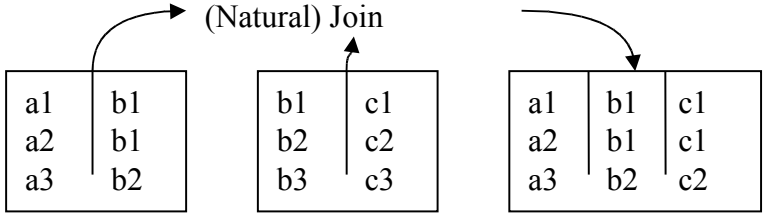



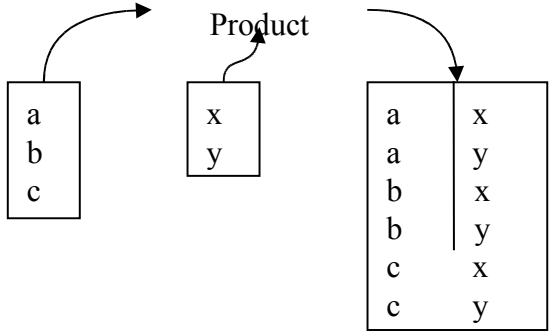



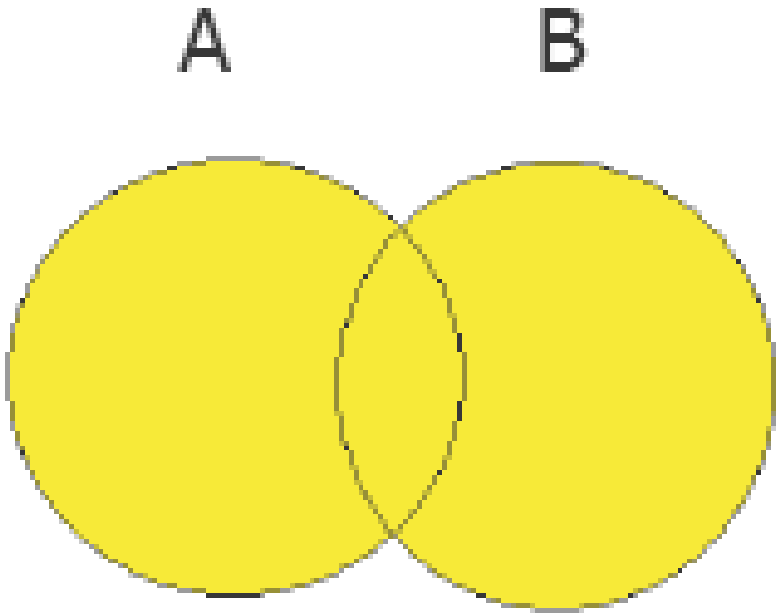



# Union

- The union of two relations produces a new relation with the all of the rows from either relation
- No duplicates are allowed (duplicates are removed)
- Common notations:
  - $R \cup S$
  - $R + S$
  - R UNION S

# Union




$A \cup B$

# Union Compatible

- To create a union of two relations, they must be "**Union Compatible**":
  - Each relation has the same number of attributes
  - The attributes must be in the same order in both relations
  - corresponding attributes must come from the same domain

# Union Compatible Example

EMP(Emp#, Name, Salary)
DEPT(Dept#, Name, Income)

- Does EMP + DEPT make sense?

Note: Most databases would let you do this!

# Union is commutative

- The union operation is commutative

$$R \cup S = S \cup R$$

**Note**: All RA operations are not commutative!

# Union - Example

BuildEmployees

| EMP # | Name | Address | DoB | NID |
|---|---|---|---|---|
| 1 | Bahadur | Dis2 | 12/3/54 | 12345 |
| 2 | Kabir | Dis4 | 3/1/49 | 23456 |
| 4 | Wasil | Dis6 | 4/7/53 | 34567 |
| 5 | Edris | Dis2 | 9/6/45 | 45678 |
| 7 | Laila | Dis9 | 12/2/39 | 56789 |
| 8 | Chamtal | Dis1 | 4/9/55 | 67890 |

# Union - Example

DesignEmployees

| EMP # | Name | Address | DoB | NID |
|---|---|---|---|---|
| 1 | Bahadur | Dis2 | 12/3/54 | 12345 |
| 3 | Adalat | Dis11 | 5/11/44 | 98765 |
| 5 | Edris | Dis2 | 9/6/45 | 45678 |
| 7 | Laila | Dis9 | 12/2/39 | 56789 |
| 9 | Jaihona | Dis2 | 6/7/42 | 87654 |

# Union – Example

What is the result of

BuildEmployees UNION DesignEmployees?

# Union - Example

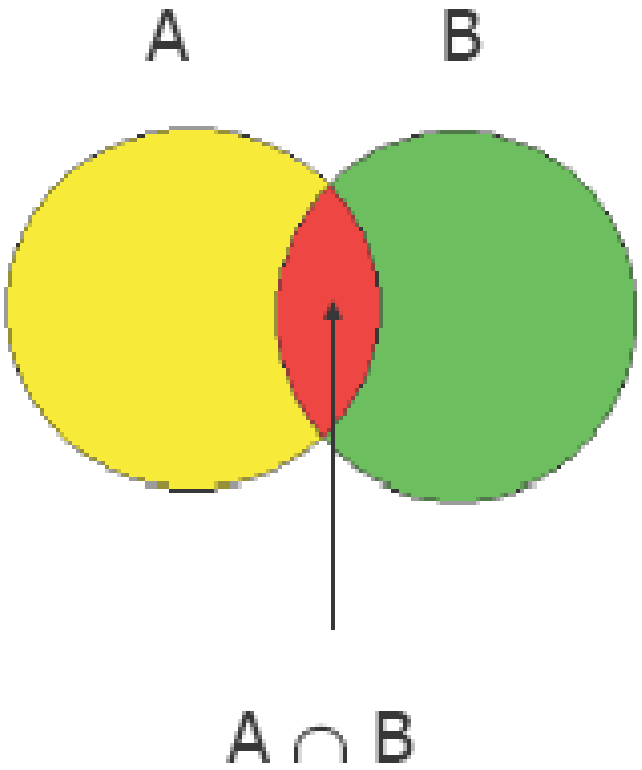
Result:

| EMP# | Name | Address | DoB | NID |
|---|---|---|---|---|
| 1 | Bahadur | Dis2 | 12/3/54 | 12345 |
| 2 | Kabir | Dis4 | 3/1/49 | 23456 |
| 3 | Adalat | Dis11 | 5/11/44 | 98765 |
| 4 | Wasil | Dis6 | 4/7/53 | 34567 |
| 5 | Edris | Dis2 | 9/6/45 | 45678 |
| 7 | Laila | Dis9 | 12/2/39 | 56789 |
| 8 | Chamtal | Dis1 | 4/9/55 | 67890 |
| 9 | Jaihona | Dis2 | 6/7/42 | 87654 |

# Intersection

- The intersection of two relations is a new relation that contains the rows that appear in both relations
- The relations must be union compatible
- Notation:
  - $R \cap S$
  - R INTERSECT S

# Intersection



# Intersection is commutative

- Order of relations in the operation is irrelevant – the result will be the same as

$$R \cap S = S \cap R$$

# Intersection- Example

BuildEmployees

| EMP # | Name | Address | DoB | NID |
|---|---|---|---|---|
| 1 | Bahadur | Dis2 | 12/3/54 | 12345 |
| 2 | Kabir | Dis4 | 3/1/49 | 23456 |
| 4 | Wasil | Dis6 | 4/7/53 | 34567 |
| 5 | Edris | Dis2 | 9/6/45 | 45678 |
| 7 | Laila | Dis9 | 12/2/39 | 56789 |
| 8 | Chamtal | Dis1 | 4/9/55 | 67890 |

# Intersection- Example

DesignEmployees

| EMP # | Name | Address | DoB | NID |
|---|---|---|---|---|
| 1 | Bahadur | Dis2 | 12/3/54 | 12345 |
| 3 | Adalat | Dis11 | 5/11/44 | 98765 |
| 5 | Edris | Dis2 | 9/6/45 | 45678 |
| 7 | Laila | Dis9 | 12/2/39 | 56789 |
| 9 | Jaihona | Dis2 | 6/7/42 | 87654 |

# Intersection – Example

What is the result of

BuildEmployees INTERSECT DesignEmployees?

# Intersection- Example

Results:

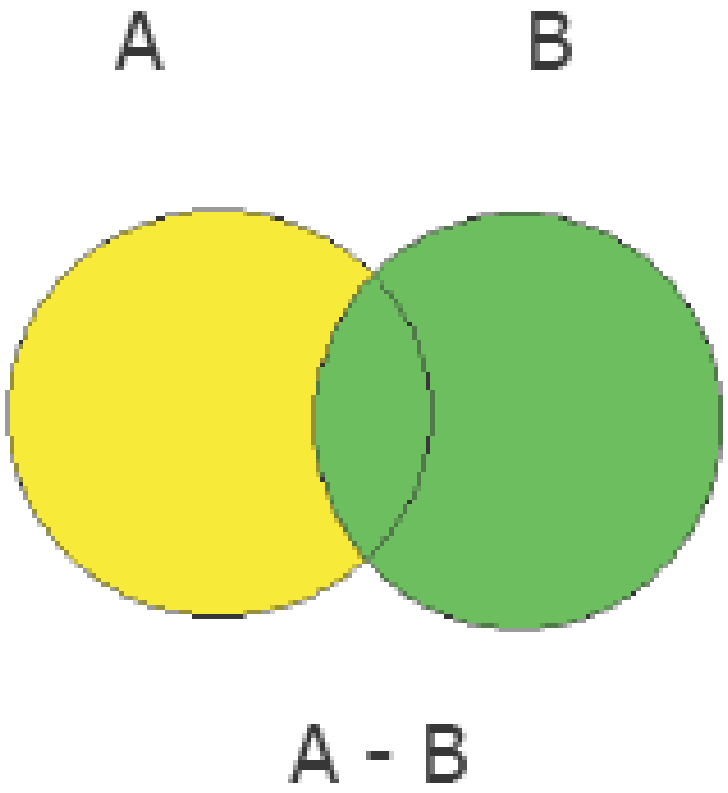
| EMP # | Name | Address | DoB | NID |
|---|---|---|---|---|
| 1 | Bahadur | Dis2 | 12/3/54 | 12345 |
| 5 | Edris | Dis2 | 9/6/45 | 45678 |
| 7 | Laila | Dis9 | 12/2/39 | 56789 |

# Difference

- Difference gives all of the rows from the first relation that do not appear in the second relation

- Relations must be union compatible
- Notations:
  R – S
  R DIFFERENCE S

# Difference



# Difference is not commutative

- Order does change the resulting relation, just as order is important in subtraction operation in math:

$$R - S \neq S - R$$

# Difference - Example

BuildEmployees

| EMP # | Name | Address | DoB | NID |
|---|---|---|---|---|
| 1 | Bahadur | Dis2 | 12/3/54 | 12345 |
| 2 | Kabir | Dis4 | 3/1/49 | 23456 |
| 4 | Wasil | Dis6 | 4/7/53 | 34567 |
| 5 | Edris | Dis2 | 9/6/45 | 45678 |
| 7 | Laila | Dis9 | 12/2/39 | 56789 |
| 8 | Chamtal | Dis1 | 4/9/55 | 67890 |

# Difference - Example

DesignEmployees

| EMP # | Name | Address | DoB | NID |
|---|---|---|---|---|
| 1 | Bahadur | Dis2 | 12/3/54 | 12345 |
| 3 | Adalat | Dis11 | 5/11/44 | 98765 |
| 5 | Edris | Dis2 | 9/6/45 | 45678 |
| 7 | Laila | Dis9 | 12/2/39 | 56789 |
| 9 | Jaihona | Dis2 | 6/7/42 | 87654 |

# Difference – Example

What is the result of:

DesignEmployees - BuildEmployees?

# Difference - Example

Results:

| EMP # | Name | Address | DoB | NID |
|---|---|---|---|---|
| 3 | Adalat | Dis11 | 5/11/44 | 98765 |
| 9 | Jaihona | Dis2 | 6/7/42 | 87654 |

# Selection

- Selection produces a new relation with a subset of the original relation rows
- Notation:
  - RELATION WHERE condition
    - EMP WHERE Dept# = 3
  - Condition can be: =, >, <, ≤,≥,
    OR: V,  AND

# Selection - Example

| Dep# | DepName |
|---|---|
| 1 | Sys Admin |
| 2 | Programming |
| 3 | Finance |
| 4 | Management |
| 5 | Planning |

DEPT WHERE Dep# = 1 OR Dep# = 3 OR Dep# = 5

What result is expected? *Next Slide*

# Selection - Example

Results:

| Dep# | DepName |
|---|---|
| 1 | Sys Admin |
| 3 | Finance |
| 5 | Planning |

# Projection

- The project of a relation produces a new relation with selected columns from the original relation (i.e. limits the attributes that will appear in the new relation)
- Can also use to rearrange the order of columns in a relation
- Duplicate rows are removed during a projection
- Notation:
  - RELATION[attr1, attr2, attr4]

# Projection- Example

BuildEmployees

| EMP # | Name | Address | DoB | NID |
|---|---|---|---|---|
| 1 | Bahadur | Dis2 | 12/3/54 | 12345 |
| 2 | Kabir | Dis4 | 3/1/49 | 23456 |
| 4 | Wasil | Dis6 | 4/7/53 | 34567 |
| 5 | Edris | Dis2 | 9/6/45 | 45678 |

What is the result of: BuildEployees [Emp#, Name, DoB]

# Projection- Example

Results: BuildEmployee [EMP#, Name, DoB]

| EMP# | Name | DoB |
|---|---|---|
| 1 | Bahadur | 12/3/54 |
| 2 | Kabir | 3/1/49 |
| 4 | Wasil | 4/7/53 |
| 5 | Edris | 9/6/45 |

# Relational Algebra (RA)
# 03

By: M Shuaib Zarinkhail 2010

# Product

- The set of rows formed by combining each row in one relation with each of the rows in another relation
- Also called:
  - Cartesian product

  or
  - Cross product
- Notations: A x B, A PRODUCT B

# Product - Example

EMP

| EMP# | Name | Dept# |
|---|---|---|
| 1 | Bahadur | 1 |
| 2 | Kabir | 2 |
| 3 | Adalat | 2 |
| 4 | Wasil | 3 |
| 5 | Edris | 1 |

DEP

| Dep# | DepName |
|---|---|
| 1 | Sys Admin |
| 2 | Programming |
| 3 | Finance |
| 4 | Management |
| 5 | Planning |

What is the result of EMP X DEP?

| EMP# | Name | EMP.Dept# | DEPT. Dept# | DeptName |
|---|---|---|---|---|
| 1 | Bahadur | 1 | 1 | Sys Admin |
| 1 | Bahadur | 1 | 2 | Programming |
| 1 | Bahadur | 1 | 3 | Finance |
| 1 | Bahadur | 1 | 4 | Management |
| 1 | Bahadur | 1 | 5 | Planning |
| 15 | Rows | removed for | readability | |
| 5 | Edris | 1 | 1 | Sys Admin |
| 5 | Edris | 1 | 2 | Programming |
| 5 | Edris | 1 | 3 | Finance |
| 5 | Edris | 1 | 4 | Management |
| 5 | Edris | 1 | 5 | Planning |

# Division

- The division is a binary operation
- The result consists of the restrictions of tuples in *R* to the attribute names unique to *R*
  - i.e. in the header of *R* but not in the header of *S*, for which it holds that all their combinations with tuples in *S* are present in *R*
  - For an example see the tables: "*Completed*", "*DBProject*" and their division in NEXT SLIDE

# Division

- Notation: $R \div S$

Note: Attributes of S should be a subset of R, if not, the result will be an empty set

# Division Example

| Completed | |
|---|---|
| **Student** | **Task** |
| Farid | Database1 |
| Farid | Database2 |
| Farid | Compiler1 |
| Fawad | Database1 |
| Fawad | Compiler1 |
| Sara | Database1 |
| Sara | Database2 |

| DBProject |
|---|
| **Task** |
| Database1 |
| Database2 |

| Completed ÷ DBProject |
|---|
| **Student** |
| Farid |
| Sara |

# Joins

- Need a way to “connect” two or more relations with each other
- The general term used for this operation is a “join”
- To join two relations, we need attributes in each relation that have the same domain
- A join can return an empty relation

# Join – Example

STUDENT (NID, Name, Major, LocalAddress, GPA)

DB409 (*NID*, Section, FinalGrade)

In this case, the NID (National Identification) uses the same domain in each relation, so these two relations could be joined

# Joins Types

- Inner joins
  - Equijoin
  - Natural join
- Outer joins
  - Left outer join
  - Right outer join
  - Full outer join

# Equijoin

- In an equijoin, two attributes are joined using an equality test
  - Both attributes appear in final result
- Example:

**STUDENT** JOIN (SID = StudentNumber) **ENROLLMENT**

# Natural join

- Similar to an equijoin
- Two attributes are joined using an equality test
  - Only one of the joining attributes appears in final result

# Equijoin- Example

EMP

| Emp# | Name | Dept# |
|---|---|---|
| 1 | Bahadur | 1 |
| 2 | Kabir | 2 |
| 3 | Adalat | 2 |
| 4 | Wasil | 3 |
| 5 | Edris | 1 |

DEP

| Dept# | DeptName |
|---|---|
| 1 | Sys Admin |
| 2 | Programming |
| 3 | Finance |
| 4 | Management |
| 5 | Planning |

| Emp# | Name | EMP.Dept# | DEP.Dept# | |
|---|---|---|---|---|
| 1 | Bahadur | 1 | 1 | Sys Admin |
| 2 | Kabir | 2 | 2 | Programming |
| 3 | Adalat | 2 | 2 | Programming |
| 4 | Wasil | 3 | 3 | Finance |
| 5 | Edris | 1 | 1 | Sys Admin |

**EMP** JOIN (EMP.Dept# = DEPT.Dept#) **DEPT**

# Natural Join – Example

EMP

| Emp# | Name | Dept# |
|---|---|---|
| 1 | Bahadur | 1 |
| 2 | Kabir | 2 |
| 3 | Adalat | 2 |
| 4 | Wasil | 3 |
| 5 | Edris | 1 |

DEP

| Dept# | DeptName |
|---|---|
| 1 | Sys Admin |
| 2 | Programming |
| 3 | Finance |
| 4 | Management |
| 5 | Planning |

| Emp# | Name | Dept# | |
|---|---|---|---|
| 1 | Bahadur | 1 | Sys Admin |
| 2 | Kabir | 2 | Programming |
| 3 | Adalat | 2 | Programming |
| 4 | Wasil | 3 | Finance |
| 5 | Edris | 1 | Sys Admin |

**EMP** JOIN (EMP.Dept# = DEPT.Dept#) **DEPT**

# Joins

- You can use other comparisons aside from equality when doing joins

- Example:

EMP JOIN (EMP.Dept# < DEPT.Dept#) DEPT

What results would you expect?

| Emp# | Name | EMP.Dept# | DEP.Dept# | |
|---|---|---|---|---|
| 1 | Bahadur | 1 | 2 | Programming |
| 1 | Bahadur | 1 | 3 | Finance |
| 1 | Bahadur | 1 | 4 | Management |
| 1 | Bahadur | 1 | 5 | Planning |
| 2 | Kabir | 2 | 3 | Finance |
| 2 | Kabir | 2 | 4 | Management |
| 2 | Kabir | 2 | 5 | Planning |
| 3 | Adalat | 2 | 3 | Finance |
| 3 | Adalat | 2 | 4 | Management |
| 3 | Adalat | 2 | 5 | Planning |
| 4 | Wasil | 3 | 4 | Management |
| 4 | Wasil | 3 | 5 | Planning |
| 5 | Edris | 1 | 1 | Sys Admin |
| 5 | Edris | 1 | 2 | Programming |
| 5 | Edris | 1 | 4 | Management |
| 5 | Edris | 1 | 5 | Planning |

# Outer joins

- There are cases where you want
  - All entries in one table
  - Any matching entries in the other table
- Outer-joins allow you to do this
- Similar notation:

**EMP** LEFT OUTER JOIN
(EMP.dept# = DEPT.Dept#) **DEPT**

# Left outer join

- Gets
  - All rows from the left hand relation
  - Any matching rows from the right hand relation
- In case of no matching rows, leave columns as NULL

# Left Outer Join – Example

DEP

| Dept# | DeptName |
|---|---|
| 1 | Sys Admin |
| 2 | Programming |
| 3 | Finance |
| 4 | Management |
| 5 | Planning |

EMP

| Emp# | Name | Dept# |
|---|---|---|
| 1 | Bahadur | 1 |
| 2 | Kabir | 2 |
| 3 | Adalat | 2 |
| 4 | Wasil | 3 |
| 5 | Edris | 1 |

# Left Outer Join – Example

**DEPT** LEFT OUTER JOIN (DEPT.Dept# = EMP.Dept#) **EMP**

| Dept# | DeptName | Emp# | Name | Dept# |
|---|---|---|---|---|
| 1 | Sys Admin | 1 | Bahadur | 1 |
| 1 | Sys Admin | 5 | Edris | 1 |
| 2 | Programming | 2 | Kabir | 2 |
| 2 | Programming | 3 | Adalat | 2 |
| 3 | Finance | 4 | Wasil | 3 |
| 4 | Management | Null | Null | Null |
| 5 | Planning | Null | Null | Null |

# Null Values

- Null means that there is no data stored in the database for that particular instance
- Null is not equal to zero
  - Null ≠ 0
- Null is not equal to empty string
  - Null ≠ “” (Empty string)
- Null is equal to unknown data value
  - Null = Unknown

# Right outer join

- Gets
  - All rows from the right hand relation
  - Any matching rows from the left hand relation
- In case of no matching rows, leave columns as NULL

# Right outer join example

STUDENT

| StudentID | Name | Major |
|---|---|---|
| 123 | Billal | IT |
| 234 | Samira | CS |
| 345 | Timor | SE |
| 456 | Adel | BUS |

DEPT

| Dept | Location | Students |
|---|---|---|
| IT | Bldg 70 | 1250 |
| CS | Bldg 10 | 700 |
| SE | Bldg 09 | 500 |
| HIS | Bldg 13 | 67 |
| ART | Bldg 56 | 70 |

**STUDENT** RIGHT OUTER JOIN (Major = Dept) **DEPT**

What is the result of this join?

# Right outer join results

| StudentID | Name | Major | Dept | Location | Students |
|---|---|---|---|---|---|
| 123 | Billal | IT | IT | Bldg 70 | 1250 |
| 234 | Samira | CS | CS | Bldg 10 | 700 |
| 345 | Timor | SE | SE | Bldg 09 | 500 |
| Null | Null | Null | HIS | Bldg 13 | 67 |
| Null | Null | Null | ART | Bldg 56 | 70 |

# Full outer join

- Combines the left and right outer joins
  - You see all rows from both relations
  - Where the rows match the join criteria, both row's values will appear in the new row
  - Nulls will appear in case where the join criteria are not met

# Full outer join example

STUDENT

| StudentID | Name | Major |
|---|---|---|
| 123 | Billal | IT |
| 234 | Samira | CS |
| 345 | Timor | SE |
| 456 | Adel | BUS |

DEPT

| Dept | Location | Students |
|---|---|---|
| IT | Bldg 70 | 1250 |
| CS | Bldg 10 | 700 |
| SE | Bldg 09 | 500 |
| HIS | Bldg 13 | 67 |
| ART | Bldg 56 | 70 |

# Full outer join results

| StudentID | Name | Major | Dept | Location | Students |
|---|---|---|---|---|---|
| 123 | Billal | IT | IT | Bldg 70 | 1250 |
| 234 | Samira | CS | CS | Bldg 10 | 700 |
| 345 | Timor | SE | SE | Bldg 09 | 500 |
| Null | Null | Null | HIS | Bldg 13 | 67 |
| Null | Null | Null | ART | Bldg 56 | 70 |
| 456 | Adel | Bus | Null | Null | Null |

# Combining Operations

- Different operations can be combined in an expression
  - Example:

    Get the names of all departments with no student records
- What operations do you need to get the results?

# Combining operations

Join

**DEPT** LEFT OUTER JOIN(Dept = Major) **STUDENT**

WHERE StudentID = Null [Dept]

Selection

Projection

# SQL

- Structured Query Language (SQL) is based on relational algebra
- Queries will use combinations of the operations we saw in the last two lectures

# Class Exercise

- Installing MySQL DBMS to the lab machines or your own PCs