University of Warwick
Department of Computer Science

CS258

Database Systems

Cem Yilmaz

August 16, 2023

Contents

1 Introduction to Data

Data comes into two different kinds: structured and unstructured.

- Structured - e.g., tables with predefined columns
- Unstructured - e.g., web pages, text docs, images, videos...

Semi-structured data also exists and is found within XML and JSON. Traditionally, these are structured databases and were defined to be so.

Definition 1.1. Relation

Informally, a relation is a table of values having:

- A set of rows. The data elements in each row represent certain facts that correspond to a real-world entity or relationship. In the formal modal, rows are called tuples.
- Each column represents a characteristic / attribute of interest of that entity. Has a column header that gives an indication of the meaning of the data items in that column. In formal model, column header is called an attribute name.

2 Relational Model

2.1 Key

Each row must be uniquely identifiable in the table. The key of the row does this. Sometimes row-ids or sequential numbers are assigned as keys to identify the rows in a table. This is called an artificial key or surrogate key.

2.2 Table

A table is just an acceptable visual representation of the mathematical notion of relation. To formulate queries, we specify that table name(s), and attributes names of interest and special constraints (aka predicates) that need to be satisfied in order for a data item (=row) to be of interest.

2.3 Formal Definition

The schema of a relation is denoted by

$$R(A_1, A_2, \ldots, A_n)$$

where R is the name of the relation
The attributes / columns of the relations are denoted by $A_1, A_2, \ldots, A_n$. For example,

$$CUSTOMER(CUST-ID, CUST-NAME, ADDRESS, PHONENO)$$

Each Attribute/column has a domain or a set of valid values. For example, CUST-ID is a 7 digit number. More about this is discussed in section 2.10.
A tuple (aka row) of a relation is an ordered set of values enclosed in angled brackets $< \ldots >$. Each value is derived from an appropriate domain. For example,

$$< 632895, \text{"PETER T."}, \text{"2 Main St. Warwick"}, \text{"(024)894}-\text{2000"} >$$

A domain has a logical definition in the real world. A domain also has data format. For example, USA_phone_numbers may have a format: $(ddd)ddd - dddd$ where each d is a decimal digit. The attribute name designates thee role played by a domain in a relation. Used to interpret the meaning of the data elements corresponding to that attribute. Example, the domain Date may be used to define two attributes named "invoice-date" and "payment-date" with different meanings.

Definition 2.1. Relation State

The relation state R is the set that contains all the set of tuples in the relation. The relation state is a subset of the Cartesian product of the domains of its attributes. To put it all together,

$$
\begin{aligned}
R(A_1, A_2, A\ldots, A_n) &\text{ is the schema of the relation} \\
R &\text{ is the name of the relation} \\
A_1, A_2, \ldots, A_n &\text{ are the attributes (columns) of the relation} \\
r(R) &\text{ is a specific state or instance of relation} \\
R &\text{ is an actual set of tuples (rows)} \\
r(R) = \{t_1, t_2, \ldots, t_n\} &\text{ where each } t_i \text{ is an } n\text{−tuple} \\
t_i =< v_1, v_2, \ldots, v_n > &\text{ where each } v_j \text{ comes from } dom(A_j) \\
r(R) \subset dom(A_1) \times dom(A_2) \times \ldots \times dom(A_n)
\end{aligned}
$$

To create a table, we would write

```sql
CREATE TABLE Drinkers (
  name CHAR(31) Primary Key,
  Addr CHAR(50) DEFAULT '123 Sesame St',
  phone CHAR(16) NOT NULL
);
```

Listing 1: Creating a table

The constraints include the fact that there are key constraints, entity integrity constraints and referential integrity constraints. Also there exist domain constraints, values in an attribute in a tuple must come from the domain of that attribute. Values could also be NULL if allowed.

2.4 Key Constraints

Superkey of R : a subset of attributes of R, SK, such that:
In any valid state for $r(R)$: for all two distinct tuples t_1 and $t_2 \in r(R)$, where $t_1[SK] \neq t_2[SK]$, where SK is the superkey. For example, let us define superkey SK with the attributes

$$SK = \{\text{Country, PhoneNumber}\}$$

then, no two entries country and phone number coincides.
A candidate key of R is a minimal super for any key K if:

- The removal of any attribute from K results in a set of attributes that is no longer a super key.

Definition 2.2. Superkey

Superkey is for all two tuples t_n and t_k in $r(R)$, there does not exist $t_n[SK] = t_k[SK]$.

Definition 2.3. Candidate Key

Otherwise known as the minimal key, it is by definition a super key that for which, if any attribute is removed from its definition of SK, it is no longer a super key.

2.5 Entity Integrity Constraints

The primary key PK cannot be NULL in any tuple of $r(R)$. This is because primary key values are used to identify the individual tuples. If PK has several attributes, null is not allowed as a value in any of these attributes. Also, any attribute of R even non-key may not be allowed to be NULL.

2.6 Referential Integrity Concerns cross-table relationships

A table students lists data about students, such as age, grades, etc. Students has an attribute STUDENTID, and suppose it is the primary key. Then, another table student-courses has also an attribute STUDENT ID, listing, for each module, which student are taking it, at which term etc. One could argue that all this info should be in one table, however, that is not a good idea and will be explained later. A logical DB Design spreads this information across tow tables, it is fundamental to R-DBMS design. Tables are "linked" according to references between them. Care must be exercised which such cross-table relationships are exercise.d.

2.7 Referential Integrity Constraints

A set of attributes FK from relation R_1 is a foreign key references relation R_2 : Attributes in the FK from R_1 have the same domain as the attributes of in the primary key PK of R_2, and a value of FK in a tuple t_1 of R_1 must either:

- refer to a value of the PK of some tuple t_2 in R_2,
- be NULL

Formally,

- t_1 in R_1 references t_2 in R_2 if $t_1[FK] = t_2[PK]$ or
- t_1 in R_1 makes no reference if $t_1[FK] = NULL$

Table 1: R_1

PK	FK
1	1
2	NULL
3	2

Table 2: R_2

PK	data
1	alpha
2	beta
3	gamma

Where in this case. $1 \to 1$, $2 \to 2$ from R_1 FK to R_2 PK. All of these constraints are expressed in create table in SQL. There also exist "semantic attribute integrity constraints" where, for example, things like bank balance > 0 are expressed.

2.8 Possible Violations for each operation

- Domain - one of the attribute values for new tuple is not in the attribute domain
- Key - the value of a key attribute in new tuple already exists
- Referential integrity - a foreign key value in new tuple references a primary key value that does not exist in referenced relation
- Entity integrity - if primary key value is null in new tuple.
- Delete - may violate only referential integrity, if primary key value of the tuple being deleted is referenced from other tuples in other relations. Some option must be specified during database design for each foreign key constraint on how to handle such deletions leading to relational integrity violations.
- UPDATE - obviously the domain and NOT NULL constraints may be violated on an attribute being modified. Furthermore, updating the primary key duplicates, updating a foreign key may violate referential integrity. Updating an ordinary attribute (not PK or FK) can violate only domain constraints.

2.9 Functional Dependencies

We need functional dependencies as they are a formal tool that allow us derivation of 'good' DB designs (relations and their schemata). Ultimately, a good design depends on the dependencies between attributes between to be in the same relation. Assume, X, Y, Z represent sets of attributes. A, B, C represent single attributes. Notation ABC implies $\{A, B, C\}$. $X \to Y$ is an assertion about a relation R and two sets of attributes from R.
If $X \to Y$ the values of the Y component of a tuple depend on values of the X component. i.e., the values of the X component of a tuple uniquely (functionally) determine those of the Y component.
So, $X \to Y$ specifies that whenever two tuples R agree on values of all the attributes of X, they must also agree on values of attribute of Y. Formally, $t_1[X] = t_2[X] \to t_1[Y] = t_2[Y]$.

2.10 Schemas

A db contains one or more schemas. Each schema contains one or more tables. To Create a schema, we type

CREATE SCHEMA company AUTHORIZATION 'Jsmith';

Creating a schema "company" owned by "Jsmith". We can create tables for each schema. The command is

CREATE TABLE

which creates a relation R. However, we must also specify the name of the table, its attributes (columns) along with their types, and any constraints. All created tables belong in a schema and belong to a special "public" schema if no other has been created. Relations and its tuples are actually created and stored as a file by the DBMS. Virtual relations/tables which are created using VIEW do not correspond to a physical file. The attribute/constraint format for a relation R is as the following:

$(<$ Att1Name $><$ Att1DataType $><$ AttConstraints $>, <$ AttName2 $><$ Att2DataType $><$ AttConstraints $>, \ldots)$

Example 2.1.

CREATE TABLE *students*(studentID INTEGER PRIMARY KEY, studentName VARCHAR(30), courseID INTEGER);

2.11 Constraints

Some basic constraints include:

- Key constraint: A primary key is unique
- Entity integrity: a primary key value cannot be null
- Referential integrity constraints: a "foreign key" must have a value that is already present as a primary key, or it may be null
- Plus attribute constraints

Additional restrictions include NOT NULL, default value of an attribute i.e., DEFAULT$<$ value $>$, CHECK i.e., CHECK(Age > 0 AND Age < 125), PRIMARY KEY, UNIQUE. The foreign key constraint, as we know, must have that it is a subset of the primary key it is referencing to. As such, with updates and deletions, we may have errors. To remedy this, we have the commands

ON DELETE and ON UPDATE

which can be paired with the commands

DEFAULT, CASCADE, SET NULL

Cascade refers to the deletion of child tables foreign keys where $FK = PK$.

3 Queries

3.1 Basic Queries

A query is done using the command

$$\text{SELECT } < \text{attribute1}, \ldots \text{attributei} > \text{ FROM } < \text{table list} > \text{ WHERE } < \text{condition} >$$

One can use the special operator $*$ in the attribute which retrieves all attributes available. If there are tables with the same name, then we can specify the table and attribute by adding the table name with a dot before the attribute. For example,

SELECT students.studentID from students;

You can further alias names of tables or even attributes/columns, for example, below we will rename the table

SELECT s.studentID FROM students AS S;

which means that the name S is now used for any qualification for the table name (including in WHERE clauses). The logical comparison operators that exist are:

$$=, <, <=, >, >=, <>$$

However, when using $=$ to filter by strings, it is important to put the string around $'$. One could also use

BETWEEN low-value AND high-value

When querying, SQL first uses the FROM clause and selects the table. Then, it applies WHERE to find the correct rows in the table. Finally, it applies SELECT to filter the columns/attributes.

3.1.1 Ordering

You can further order your final result by adding

ORDER BY attribute(s) [ASC|DESC]

in the end of your query. You can also mathematical expressions within the query. You could, for example, retrieve the price in SELECT price*147 which would multiply all prices by 147.

3.1.2 Combining

You can further complicate and define new columns, such as

SELECT drinker, 'likes Earl Grey' AS whoLikesEarlGrey FROM Likes WHERE tea = 'Earl Grey';

This would define a new column whoLikesEarlGrey with the string 'likes Early Grey' for all rows. You could create a more sophisticated column by placing functions such as string count for the drinker name.

3.2 More Complicated Queries

3.2.1 String Matching

the WHERE clause can have conditions for strings, in which a string is compared with a pattern to see if it matches. The general form is

$$< \text{StringAttribute} > \text{ LIKE } < \text{pattern} >$$
$$< \text{StringAttribute} > \text{ NOT LIKE } < \text{pattern} >$$

The pattern is a quoted string with special characters where

$$\% = \text{ "any string" of zero or more characters}$$
$$\_ = \text{ "any character", exactly one character}$$

For example, the query

SELECT name FROM drinkers WHERE phone LIKE '%1926______'

Would create a query that finds drinkers whose phone has the dialling code 1927. The % accommodates for possible country codes e.g. +44, +90 and the specific number of _ ensures the length of the phone has exactly 6 characters after 1926.

3.2.2 NULL

We need to be able to test if a value is NULL. The syntax is as follows:

$$< \text{Attribute} > \text{ IS (NOT) NULL}$$

The (NOT) was placed to emphasise that one could place the word NOT if desired.

4 Database Modifications

4.1 Insertion

$$\text{INSERT INTO } < \text{tablename} > \text{ VALUES } (< \text{val1} >, < \text{val2} >, \ldots < \text{valn} >);$$

Each list refers to a record (tuple) being inserted. Each value belongs to the attribute of the same position in the attribute list of the schema. You can also insert partial records by specifying which attributes you would like to insert. That is,

$$\text{INSERT INTO } < \text{tablename} >< \text{A1}, \ldots, \text{Aj} > \text{ VALUES } (< \text{val1} >, < \text{val2} >, \ldots < \text{valn} >);$$

where $< \text{A1}, \ldots, \text{Aj} >$ is a partial attribute list

4.2 Deletion

$$\text{DELETE FROM } < \text{tableName} > \text{ WHERE } < \text{expression} >;$$

For example,

$$\text{DELETE FROM students WHERE studentID} = 2000;$$

would delete the student with the studentID 2000. Note that it has to pass referential integrity checks.

4.3 Updating

$$\text{UPDATE } < \text{relation} > \text{ SET } < \text{list of (attribute assignment) pairs} > \text{ WHERE } < \text{condition on tuples} >;$$

For example,

$$\text{UPDATE drinkers SET phone} =' 09126 - 554555' \text{ WHERE name = 'Peter';}$$

5 Programming DB

5.1 JDBC: Java Data Base Connectivity

The JDBC is an API that calls SQL commands from Java. It is implemented as a driver for a particular DBMS with methods, objects and classes. One Java program can have connections to several databases and possibly even connected to different DBMSs. These are called data sources.

The driver manager class is used to handle multiple connections to different DBs i.e., managing their drivers, and mapping clients to right drivers. The methods include, but are not limited to:

- getDriver
- registerDriver
- deregisterDriver

There are several driver types for different types of support and we mostly refer to type 4 drivers, which are pure Java driver for direct-to-database. Fundamentally, there are 3 key tasks carried out with the help of important new Classes/ objects.

1. The Connection class
    - A connection object is needed per DB connection
    - Created using driverManager's getConnection()
    - Has functions to connect to specified DB sources using URLs
2. The Statement class with two subclasses
    - PreparedStatement and CallableStatement which have methods to execute an SQL command.
    - Need a statement object per SQL query / update you wish to issue
3. The ResultSet class
    - Query results are returned into objects of this type
    - It is akin to tables, where result tuples are rows and columns are attributes

5.2 Programming

5.2.1 Connection

The import the JDBC class library we write

```java
import java.sql.*
```

Listing 2: JDBC import

We then load explicitly the JDBC driver: use a generic Java function for loading class e.g.,

```java
Class.forName("oracle.jdbc.driver.OracleDriver"); // for Oracle
Class.forName("org.postgresql.Driver"); // for Postgresql
```

Listing 3: Loading JDBC drivers

This register the driver with the driver manager and makes it available to the program. We now create vars for connection credentials, that is

```java
String dbacct, passwd, ssn, name;
dbacct = readentry("Enter database account:");
passwd = readentry("Enter password:");
```

Listing 4: Creating connection credentials

Although this can also be replaced with the Scanner class to read any input from the console. Finally we use a connection object to connect to DB, that is

```java
Connection conn = DriverManager.getConnection("jdbc:oracle:oci8:" + dbacct + "/" + passwd);
```

Listing 5: Connection to DB

After connection, it might be wise to check whether the Postgres Driver can be loaded by catching. In particukar,

```java
try {
  Class.forName("org.postgresql.Driver");
}
catch (ClassNotFoundException x) {
  System.out.prinln("Driver could not be loaded");
}
```

Listing 6: Catching if driver can be loaded

Example 5.1. Connecting to a local DB server

```java
String username = "me";
String password = "mypassword";
String url = "jdbc:postgresql://127.0.0.1:5432/mydb?user=" + username + "&password=" +
    password;
Connection conn = DriverManager.getConnection(url);
```

Listing 7: connecting to a local DB server

5.2.2 Statements and Queries

We now have loaded drivers and set up the DB connection. We now use a Statement object to hold a query. We use PreparedStatement if the query will be issued many times, that is, it is prepared, checked and compiled only once to save processing. It is used as such:

```java
String stmt1 = "SELECT course, year FROM students WHERE Name = ? AND age = ?";
PreparedStatement p = conn.preparedStatement(stmt1);
p.setString(1, "John"); // says that the first ? is John as string
p.setInt(2,20); // says that the second ? is 2 as int
```

Listing 8: Statement Query

After preparation, we can now run our statement using executeQuery. For every stmt object, there are 2 functions: executeQuery which returns an object of type ResultSet for the SELECT command and executeUpdate which is for INSERT, UPDATE, DELETE, CREATE and DROP, returning an integer for the number of affected tuples. The command to run our earlier statement is

```java
ResultSet r = p.executeQuery()
```

Listing 9: executing the query

The object type of ResultSet contains the set of result. After executing r call, r refers to an initial position before the 1st tuple count in result. If r is of type ResultSet, we can scroll through the results using $r.next()$. This will return a NULL in the end as the last result. In particular

```java
while (r.next()) {
  course = r.getString(1); // The 1 refers to the attribute position within the tuple as
      defined in our query
  year = r.getInt(2); // The 2 refers to the attribute position within the tuple as defined in
      our query
  System.out.println(course + "," + year );
}
```

Listing 10: Iterating through ResultSet

5.2.3 Updates

For creating tables, inserting and deleting tuples, etc. we first need a statement object, as per queries, over an established connection conn

```java
Statement stmt = conn.createStatement();
```

Listing 11: Connection

We then use the executeUpdate method, i.e.,

```java
String sql = "CREATE TABLE Students " +
             "(ID INTEGER NOT NULL, " +
             "Name VARCHAR(25), " +
             "gpa REAL DEFAULT 0.0, " +
             "PRIMARY KEY (ID))";
stmt.executeUpdate(sql)
```

Listing 12: Table Creation

You can further do other commands, such as INSERT by modifying the sql string

```java
String sql = "INSERT INTO STAFF VALUES " + "(1234567, 'Peter')";
```

Listing 13: Insert

It is also possible to use a prepared statement for the actions described above. If done so, instead of $ResultSet\ r = p.executeQuery()$ would be $int rows = p.executeUpdate()$.

5.3 Errors

As specified before, one of the best errors we can implement is to see if the driver can be loaded earlier in the notes. However, managing errors with JDBC can get more complicated for different functions

5.3.1 Connections

When creating connections, we can check if it has been established with

```java
try {
  connection conn = DriverManager.getConnection(URLstring);
}
catch (SQLException se) {
  System.out.println("Could not open connection with connection string " + URLstring);
  se.printStackTrace();
}
```

Listing 14: Connection Error Catching

5.3.2 Statements

To catch errors using statements we use

```java
try {
  Statement stmt = conn.createStatement();
}
catch (SQLException se) {
  System.out.println("Could not create statement ");
  se.printStackTrace();
}
```

Listing 15: Statements Error Catching

5.3.3 Updating INSERT

```java
try {
  String sql = "INSERT INTO Students VALUES " + "(1234567, 'Andrew H.')";
  stmt.executeUpdate(sql);
}
catch (SQLException se) {
  System.out.println("Could not insert tuple for Andrew H.");
  se.printStackTrace();
}
```

Listing 16: Update INSERT Error Catching

5.3.4 Updating TUPLE

```
try {
  int count = stmt.executeUpdate("UPDATE Students SET Name = 'Andrew Hague' WHERE ID = 1234567"
      );
catch (SQLException se) {
  System.out.println("Could not update Andrew H.");
  se.printStackTrace();
}
```

Listing 17: Update UPDATE Error Catching

5.3.5 Querying

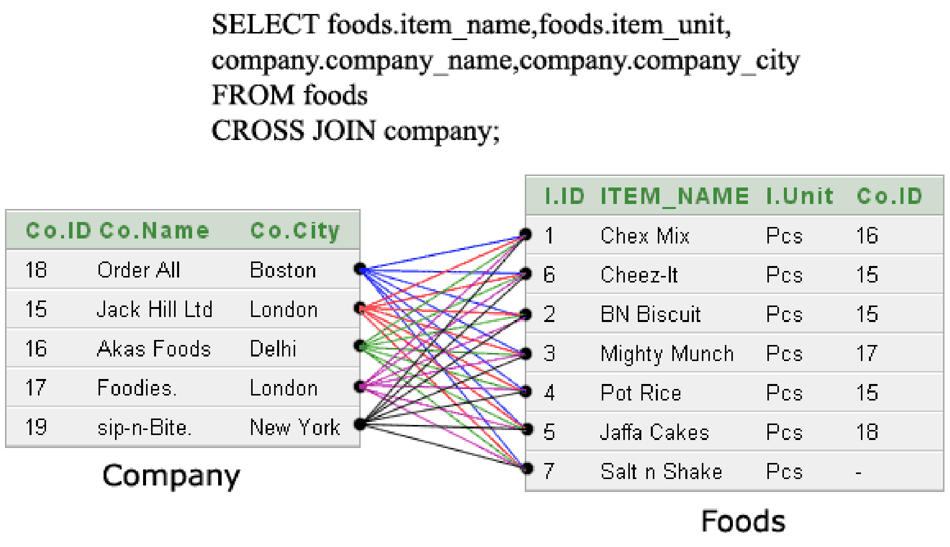
```
try {
  sql = "SELECT * FROM Students WHERE ID < 5555555";
  rs = stmt.executeQuery(query);
}
catch (SQLException se) {
  System.out.println("Could not get result of query " + sql);
  se.printStackTrace();
}
```

Listing 18: Query Error Catching

6 Advanced SQL

6.1 Multiple Tables

If we have multiple tables in FROM clause, then we compute the cartesian product of the tables, which produces a new table. And then we scan this table, one a time. You can think of this as a nested loop: for each tuple in the 1st table, concatenate it with each tuple of the 2nd table, in turn we achieve an operation called CROSS JOIN.



```
SELECT foods.item_name,foods.item_unit,
company.company_name,company.company_city
FROM foods
CROSS JOIN company;
```

Company

Co.ID	Co.Name	Co.City
18	Order All	Boston
15	Jack Hill Ltd	London
16	Akas Foods	Delhi
17	Foodies.	London
19	sip-n-Bite.	New York

Foods

I.ID	ITEM_NAME	I.Unit	Co.ID
1	Chex Mix	Pcs	16
6	Cheez-It	Pcs	15
2	BN Biscuit	Pcs	15
3	Mighty Munch	Pcs	17
4	Pot Rice	Pcs	15
5	Jaffa Cakes	Pcs	18
7	Salt n Shake	Pcs	-

Figure 1: Cross Join Table Example

Follow the coloured lines above to see how it cross joins. Each edge is a newly created tuple in the new table. To specify CROSS JOIN between two tables R and S, we write

$$R \text{ CROSS JOIN } S$$

And similarly, for NATURAL JOIN

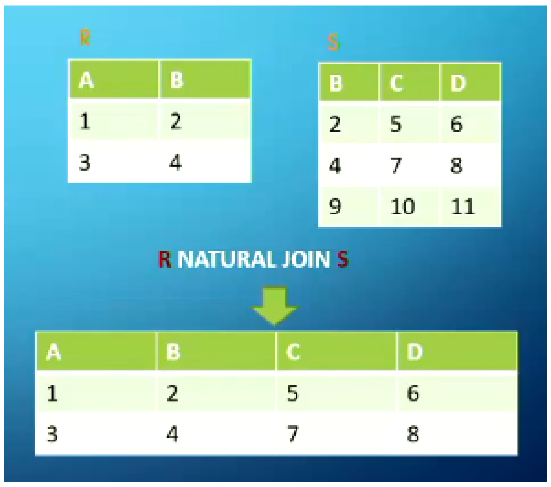
$$R \text{ NATURAL JOIN } S$$

Recall that queries create a table, therefore, we can also write

$$query_1 \text{ CROSS JOIN } query_2$$
$$query_1 \text{ NATURAL JOIN } query_2$$

6.2 NATURAL JOIN

R NATURAL JOIN S is equivalent to formatting the cross product of R and S and then looking for attributes of R that have the same name as attributes in S. Keep only tuples from product whose same-name-type attributes have same value. Only one of each set of duplicate columns is kept.



R

A	B
1	2
3	4

S

B	C	D
2	5	6
4	7	8
9	10	11

R NATURAL JOIN S

A	B	C	D
1	2	5	6
3	4	7	8

Figure 2: NATURAL JOIN

6.3 JOIN

This is an extension to natural joins and it allows us to combine constraints (in WHERE) and a product in 1 operations. The syntax is

$$R \text{ JOINS } S \text{ ON } < \text{condition} >$$

$<$ condition $>$ is used for selecting tuples. If operator is $=$ this is called an equi-join. For example,

Drinkers D JOIN Frequents F ON D.drinker = F.drinker

returns (d, a, d, b) quadruples such that drinker d lives at address a and frequents teaRoom b. The default join operator we have seen thus far is called inner join, for which there exist an explicit operator

SELECT $<$ attList $>$ FROM $<$ table1 $>$ INNER JOIN $<$ table2 $>$ ON $<$ expr $>$;

An inner join will combine the rows from $<$ table1 $>$ with the rows from $<$ table2 $>$ based on satisfying the ON boolean expression. The expression usually takes the form of a comparison between a column from table 1 and a column from table 2.

6.4 INNER JOIN

The inner join has the same functionality as

SELECT $*$ FROM student S, studentCourses C WHERE S.studentID = C.studentID

Explicitly using "Join" makes formatting more readable. Using 'ON' separates clauses that deal with sources of data i.e., belonging to FROM away from CLAUSES that deal with selection criteria

6.5 OUTER JOIN

Inner joins can "lose" information, become a tuple that doesn't join with any other relation disappears. Instead, outer join keeps this information and fills tuples that do not match with nulls if it is not matched when joining. You can further specify if you want to keep information on the left side or the right side, more specifically if you wanna keep information on the left table, then you type LEFT OUTER JOIN. You can further use the syntax $NATURAL$ to make the join natural, or use ON syntax after the OUTER JOIN to specify when to join.
SO in outer joins, some tuples that do not "join" are kept. There are variations though

- Left outer join - keep only dangling tuples from the left
- Right outer join - keep only dangling tuples from the right

For pairs that are not matched, depending on the type of outer join, the new columns are filled with NULL.

SELECT $<$ attList $>$ FROM $<$ table1 $>$ [LEFT|RIGHT] OUTER JOIN $<$ table2 $>$
ON $<$ table1.attName1 $>=<$ table2.attName2>;

You can also do a FULL outer join that would do both left and right. One can also make it a NATURAL join by adding the NATURAL word before the LEFT/RIGHT OUTER JOIN syntax.

7 Subquery

7.1 Definition

A subquery (inner query) is a parenthesised query statement following the SELECT-FROM-WHERE template and is contained in another outer query.

7.2 Scalar Subquery

You can use a subquery as a value if the result is a tuple of a single attribute. Consider the scoping rule, where an attribute refers to the most closely nested relation with that attribute.

7.3 ANY operator

the ANY(<relation>) is true if and only if it equals at least one tuple in the relation

8 Relational Algebra

8.1 Definition

Relational algebra formally defines operations on relations.

Definition 8.1. Projection

Projection is a unary operator and correspond to SELECT A in SQL, ie.,

$$\pi_A \tag{1}$$

The π is usually followed an attribute list, for example

$$\pi_{\text{Lname,Fname,Salary}}(\text{EMPLOYEE}) \tag{2}$$

Note that because this is set notation, duplicate values will be removed. Furthermore, it is illegal to project new attributes from smaller projections.

Definition 8.2. Selection

Contrary to the name, the selection in relational algebra denotes there WHERE clause in SQL. It defines the tuple of interest using a constraint c.

$$\sigma_c \tag{3}$$

The comparison operator is one of $=, \leq, <, \geq, >, \neq$ For example,

$$\sigma_{\text{Dno}=4 \text{ AND Salary}>250000}(\text{EMPLOYEE}) \tag{4}$$

It has commutative properties, i.e., you can interchange where σ locate between each other and you can sequence it which is equal to the AND.

Definition 8.3. Product

Product is a binary operator and it calculates the Cartesian product i.e.,

$$R \times S \tag{5}$$

The result of the two is

$$Q(A_1, \ldots, A_m, B_1, \ldots, B_n) \tag{6}$$

such that each tuple in Q is a combination of an m-tuple from R and n-tuple from S.

Definition 8.4. Join

The join operator corresponds to JOIN in SQL. It is denoted by

$$R \bowtie S \tag{7}$$

We can also set the join condition by

$$R \bowtie_{\text{join condition}} S \tag{8}$$

For example, you can state the join condition as SSN $=$ ESSN, and this is called an equijoin.

$$R * S \tag{9}$$

Is the natural join of R and S if they have a common attribute.

Definition 8.5. Set Operators

The usual set operators $\cup, \cap, -$ all correspond to union, intersect and except respectively

Definition 8.6. Renaming

To rename a relation or an attribute we use the

$$\rho \tag{10}$$

operator.

$$\rho_S(R)\ S \text{ is new relation name} \tag{11}$$
$$\rho_{(B_1,\ldots,B_n)}(R)\ B_i \text{ are the new attribute names} \tag{12}$$
$$\rho_{S(B_1,\ldots,B_n)}(R) \text{ Renaming both relation and attributes} \tag{13}$$

Throughout the rest of this section, we will use the following database

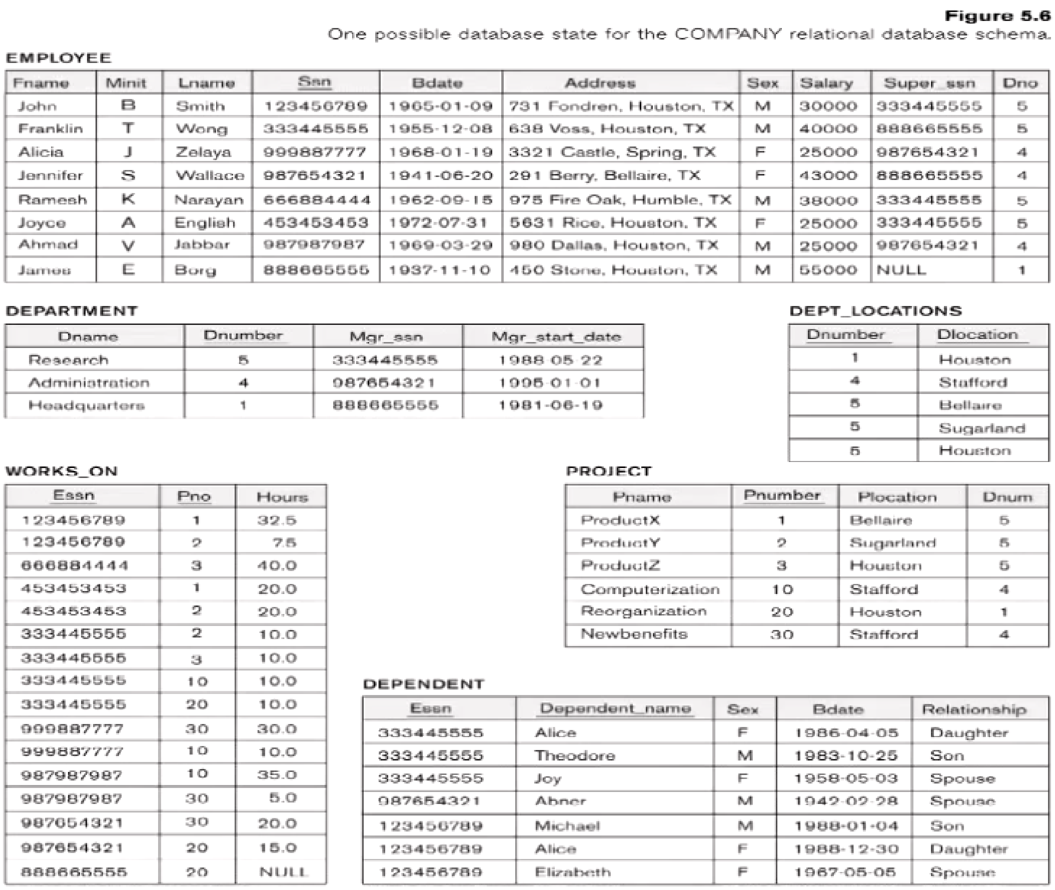


Figure 5.6
One possible database state for the COMPANY relational database schema.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Figure 3: Database

Definition 8.7. Compatability

Two relations $R(A_1, \ldots, A_n)$ and $S(B_1, \ldots, B_n)$ are type-compatible if $m = n$ and $dom(A_i) = dom(B_i)$ for $1 \leq i \leq n$. It is required that for set operations, that they are type compatible.

Definition 8.8. Completeness

For expressively, we need not consider all of the operations. Some are included for convenience or brevity. For example, the set

$$\{\sigma, \pi, \rho, \cup, -, \times\} \tag{14}$$

Definition 8.9. Division

For some relations R, S and T that hold

$$T \times S = R \tag{15}$$

We have

$$R \div S = T \tag{16}$$
$$R \div T = S \tag{17}$$

Note that if R has extra few tuples, then division is still impossible, but we just ignore those.

Definition 8.10. Aggregate Function

Group tuples by values of some attributes and apply functions separately to each group

$$_{\text{grouping attributes}}\mathfrak{I}_{\text{function list}}(R) \tag{18}$$

Where function is typically one of SUM, AVERAGE, MAXIMUM, MINIMUM, COUNT;

9 Guidelines for Data

- Guideline 1: Easy to Understand i.e., each tuple in a relation should represent one entity or relationship instance. Only refer to different entities with foreign keys.
- Guideline 2: Remove redundancy i.e., no point in storing the same name twice when we could just refer to it with a foreign key. There are consistency costs and potential anomalies caused with redundancy.
- Guideline 3: No NULLs. NULL values are harmful and should be avoided. They complicate the semantics of joins, selections and aggregation.
- Guideline 4: No spurious tuples. I.e., after decomposing a table in order to get rid of redundancy, we do not obtain any tuples that were not a part of our original table.

Function dependencies are constraints that are derived from the meaning and interrelationships of the data attributes of a relation schema R.

9.1 Implied Functional Dependencies

Given a relation schema R, the closure (cover) F^+ of a set of FDs, $F : F^+$ represents the set of all implied FDs from F. The implied functional dependencies are derived using the following properties:

Definition 9.1. Armstrong Axioms

1. Reflexive. If $Y \subseteq X$, then $X \to Y$ (called trivial FD)
2. Augmentation. If $X \to Y$, then $XZ \to YZ$
3. Transitive. If $X \to Y$ and $Y \to Z$ then $X \to Z$

9.2 Decomposition Test

In general: given R and set of FDs $F : R_1, R_2$ is a lossless decomposition of R in F if and only if:

- $R_1 \cap R_2 \to (R_1 - R_2)$
- Or
- $R_1 \cap R_2 \to (R_2 - R_1)$

Exist in F^+.
Furthermore, it is nice but hard to achieve dependency preservation.

9.3 Normalisation

Definition 9.2. First Normal Form

All attributes are atomic. They have no repeating groups and attributes can not have a substructure. Formally, the domains of attributes must be atomic. Finally, there is a key defined.

Definition 9.3. Second Normal Form

A relation is said to be 2NF if it is in 1NF and every non-key attribute is fully functionally dependent on any key and is irreducible

Definition 9.4. Third Normal Form

A relation is said to be in 3NF if it is in 2NF and no transitive functional dependency from a key to a non-key attribute

Definition 9.5. Boyce-Codd Normal Form

A relation schema R is in Boyce-Codd Normal Form if whenever an FD $X \to A$ holds in R, then X is a superkey. A 3NF relation is not in BCNF if

- There is a determinant that is NOT a superkey in a non-trivial FD
- 3NF is required that non-key attributes need to be determined only by the key
- BCNF requires additionally that the RHS of an FD must not be a prime (key) attribute unless FD is trivial

In other words, the only arrows are "out of superkeys"

10 Database Security

DBMS includes a database security and authorisation subsystem. Two types of database security mechanisms exist:

- Discretionary security mechanisms - Grant or revoke access privileges
- Mandatory security mechanisms - Enforce classification levels for users and data items

Definition 10.1. System Log

The system log includes an entry for each operation applied to the database. It is required to recover from transaction failure or system crash. For security purposes we need to trace ID information.

Definition 10.2. Discretionary Access Control

Method: Granting and revoking privileges
Implementation: supporting statements in the query language for DBA and selected users
There are two types of privileges, which is account level and relation level (for a specific table)

Definition 10.3. Access Matrix Model

The access matrix model is given by

$$M(i, j) \tag{19}$$

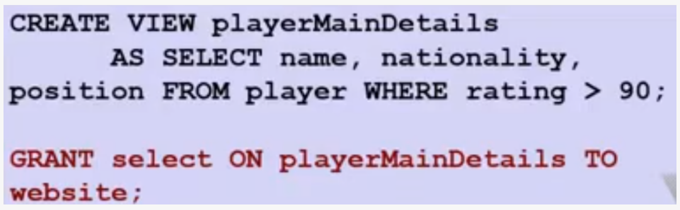
And it represents the types of privileges that a subject i holds on object j.

Definition 10.4. GRANT

SQL privileges can be granted using the GRANT command. For example,

$$\text{GRANT SELECT ON table\_name TO username;} \tag{20}$$

One may grant: SELECT, UPDATE, DELETE, INSERT. Referencing is also possible to grant. However, if one wants to limit what rows a person can view, we can create a VIEW and grant permissions for that person



```
CREATE VIEW playerMainDetails
      AS SELECT name, nationality,
position FROM player WHERE rating > 90;

GRANT select ON playerMainDetails TO
website;
```

Figure 4: View then GRANT

You can revoke grant options, and further, you can give grant option to users to give grant by adding WITH GRANT to the end of the syntax.

Definition 10.5. Mandatory Access Control

More fine-grained classifications are often required: data and users are classified into security classes. Most commercial DBMS provide discretionary access only. But DBMS vendors release special version of their RDBMSs for government,military and corporate use.