A relational database model enables logical representation of the data and its relationships.

**SLIDE 03-01**
1. A table is perceived as a 2-dimensional structure composed of rows and columns.
2. Each table row (*tuple*) represents a single entity occurrence within the entity set.
3. Each table column represents an attribute, and each column has a distinct name and datatype.
4. Each intersection of a row and column represents a single data value.
5. All values in a column must conform to the same data format (datatype).
6. Each column has a specific range of values known as the *attribute domain*.
7. The order of the rows and columns is immaterial to the DBMS.
8. Each table must have an attribute or combination of attributes that uniquely identifies each row (the *key*).

**Keys**

Keys ensure that there is precisely one way to identify a row in a table.

More formally, “Keys are one or more attributes that *determine* the other attributes in a row.”

> "Two tuples in R cannot agree on all of the attributes in set S, unless one of them is NULL. Any attempt to insert or update a tuple that violates this rule will cause the DBMS to reject the action that caused the violation." -- DSCB

Keys are also used to establish relationships among tables and to ensure data integrity.

The *Primary Key* is one or more attributes that uniquely identifies any given row (or tuple) **AND** the attribute(s) making up that Primary Key cannot be NULL.

**Functional Dependence and stuff**

The idea of “determination” is an important one in DB’s. We use it to “normalize” relations, thereby avoiding anomalies.
Functional dependence: Value of one or more attributes determines the value of one or more other attributes

- Determinant: Attribute whose value determines another
- Dependent: Attribute whose value is determined by the other attribute

**Formally:**

Let $R$ be a relation schema, and $\alpha$ and $\beta$ are sets of attributes.
The functional dependency $\alpha \rightarrow \beta$ holds on $R$ if and only if for any legal relations $r(R)$, whenever any two tuples $t_1$ and $t_2$ of $r$ agree on the attributes $\alpha$, they also agree on the attributes $\beta$. That is,

$t_1[\alpha] = t_2[\alpha] \implies t_1[\beta] = t_2[\beta]$ 

**Example:** Here’s a relation $r(A,B)$:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
</tr>
</tbody>
</table>

Here, $A \rightarrow B$ does hold, but $B \rightarrow A$ does NOT hold since $t_1=(4,1)$ and $t_2=(5,1)$, and $t_1[B] = t_2[B]$ but $t_1[A] \neq t_2[A]$

**Example:**

Here’s another example

Student (SSN, sName, address, Hscode, Hsname, Hscity, GPA, priority)

- SSN $\rightarrow$ sName
- SSN $\rightarrow$ address (assuming a student doesn’t move during enrollment)
- HScode $\rightarrow$ Hsname, HScity
- Hsname, HScity $\rightarrow$ Hscode (Assume no two HS with same name in one city)
- SSN $\rightarrow$ GPA
- GPA $\rightarrow$ priority

**SSN $\rightarrow$ GPA** (example of our old friend Transitivity!)

That’s enough about functional dependence for now. We’ll return to the topic later when we learn about Normalization.

**Back to keys**
Now we can talk about keys more specifically.

- **Composite key**: a key that is composed of more than one attribute
- **Key attribute**: Attribute that is a part of a key
- **Entity integrity**: Condition in which each row in the table has its own unique identity
  - All of the values in the primary key must be unique
  - No key attribute in the primary key can contain a null

**Other special terms:**
- **Null**: Absence of any data value that could represent:
  - An unknown attribute value
  - A known, but missing, attribute value
  - A inapplicable condition
- **Referential integrity**: Every reference to an entity instance by another entity instance is valid

Ask Students *how do you identify a key?*

There are a variety of keys we’ll talk about:

**SLIDE 03-02**

There can be LOTS of candidate keys,
Let's look at a simple DB

**SLIDE 03-03** OR MySQLWorkBench

OK, let's see what kinds of keys are in this table

**SLIDE 03-04**

- First find the functional dependencies
  - $\text{STU\_NUM} \rightarrow \text{STU\_LNAME}$
  - $(\text{STU\_FNAME}, \text{STU\_LNAME}, \text{STU\_INIT}, \text{STU\_PHONE}) \rightarrow \text{STU\_DOB, STU\_HRS, STU\_GPA}$
  - Sometimes the dependency is overspecified. Suppose we’re given:
    1. $\text{STU\_NUM} \rightarrow \text{STU\_GPA}$ and
    2. $(\text{STU\_NUM, STU\_LNAME}) \rightarrow \text{STU\_GPA}$
    Here $\text{STU\_LNAME}$ is unnecessary since we already know 1.

- **Superkey** - one or more attrs uniquely identifying a row
  - ask students
    - $\text{STU\_NUM}$
    - $(\text{STU\_FNAME, STU\_LNAME, STU\_INIT, STU\_PHONE})$

- **Candidate Key** - minimal Superkey
  - $\text{STU\_NUM}$
• Foreign Key - links to other tables
  
  o  **ask students**
  o  Can you think of other tables to which this one might be related?
  o  **Draw on board**
  o  DEPT_CODE key to DEPARTMENT table
  o  PROF_NUM key to FACULTY table
  o  ???

Note that keys must be chosen with care - what could go wrong?

• **ask students**
• Might not uniquely identify rows because of entities or data you didn't expect

**Entity & Referential Integrity**

For all this to work, one Candidate Key has to be selected for each table as its *Primary Key*.

The attribute(s) making up this key can **never** be *NULL*.

When might a *NULL* occur?

• no middle initial
• **ASK STUDENTS FOR OTHERS**

**ASK STUDENTS** why are *NULLs* problematic in a primary key?

• If it can’t uniquely identify a row/tuple
  o  How can you compare a *NULL* to other things?

**SLIDE 03-05**

*NULL* means “no value”. Not ZERO, and not the empty string “\0”

The result of any arithmetic operation involving *NULL* is *NULL*

Aggregate functions simply ignore *NULL*. For matching, *NULL* is treated like any other value, and two *NULL*s will match.

• **Entity Integrity**
  o  All primary key entries are unique and no part may be *NULL*.
  o  This ensures each row will have a unique identity and that other tables can properly reference rows via the primary keys.
  o  Example: “No invoice can have a duplicate number or be *NULL*."

• **Referential Integrity**
  o  The entry for a Foreign Key may either be *NULL* or an entry that matches a primary key in a table to which this one is related.
  o  It’s ok to have a missing reference, but it is not ok to have an invalid entry.
**ASK STUDENTS** Think about this:

**DRAW ON BOARD**

- If you have two tables linked via a Foreign Key, and you force all Foreign Keys to be non-\texttt{null} and valid, \textit{THEN} it would not be possible to ever delete a row in the table to which the Foreign key refers!
  - It is impossible to have an invalid invoice number on a Receipt.

---

**SLIDE 03-06**

Example of integrity rules

Lots of ways to handle \texttt{null}s … special values, typically.

Most DB’s allow you to specify a column as \texttt{NOT NULL}.

---

**RELATIONAL ALGEBRA**

Why do we study RA?

RA provides the formal mathematical basis for Relational DB’s.

RA is platform-independent and concise… MUCH simpler than SQL, yet complete.

Understanding RA enables the student to understand how RDBMS’s take a high-level query in \texttt{SQL} and implement it as an optimised series of sub-queries.

SQL is \textit{declarative}, describing what the user wants.

Relational Algebra is \textit{procedural}, describing the steps in how best to accomplish what the user wants.

Relational Algebra also has the property of \texttt{closure}
- Using Relational Algebra operators on existing relations produces new relations.

**Relational Set Operators**

**SLIDE 03-07**

Codd’s eight original operators:

Set operators: UNION, INTERSECTION, DIFFERENCE

Relation operators: JOIN, PROJECTION, SELECTION, CARTESIAN PRODUCT and DIVISION.

We’ll use these relations for examples.

**SLIDE 03-08**

PROJECT eliminates columns while SELECT eliminates rows.

\textit{Go through each operator, discuss, and write examples on board}

\textit{I USE SLIDES 03-09 THROUGH 03-26 TO EXPLAIN}

**DISCUSSION**

Question: Is it ever useful to compose two projection operators?

Example: \texttt{PROJECT}_{\text{cName}} (\texttt{PROJECT}_{\text{cName, pPos}} (\text{Campus}))
Question: Is it ever useful to compose two selection operators?

Example: \( \text{SELECT}_{\text{HS}>1000} \ (\text{SELECT} \ \text{yCard}='yes' \ \text{Player}) \)

*Note* Intersection doesn’t add any new expressive power to Relational Algebra since \( E_1 \ \text{INTERSECT} \ E_2 = E_1 - (E_1 - E_2) \). Explain with a Venn diagram.

**Alternate representations of Relational Algebra expressions**

Some of these expressions can get rather messy. There are a couple of schemes you can use to make them clearer.

- *expression trees* are visual aids for people, not so useful for computers.
  
  *SLIDE 03-27*

- *linear notation* allows you to break up complex expressions using temporary variables
  
  *SLIDE 03-28*

**JOINS**

Joins are extremely important to the relational model, as they are the means for combining tables in a variety of ways and subject to a variety of selection mechanisms.

Essentially, Joins allow information to be intelligently combined from two or more tables.

Joins represented by the “bowtie” symbol: \( \bowtie \).

There are quite a few of them, best explained visually using the superb diagram from C.K. Moffett. You might keep a copy handy.

*SLIDE 03-29*

- **Natural join**
  - Links tables by selecting only the rows with common values in their common attributes
  - a.k.a. common join.
  - Enforces equality on all attributes with the same name.
  - Eliminates all but one copy of duplicate named attributes

- **Equijoin**: Links tables on the basis of an equality condition that compares specified differently-named columns from each table.
  - Example: (write on board)

**Employees**

<table>
<thead>
<tr>
<th>Employee</th>
<th>Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>A</td>
</tr>
<tr>
<td>Black</td>
<td>A</td>
</tr>
<tr>
<td>Black</td>
<td>B</td>
</tr>
</tbody>
</table>
Projects

<table>
<thead>
<tr>
<th>SecLevel</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Classified</td>
</tr>
<tr>
<td>B</td>
<td>Unclassified</td>
</tr>
</tbody>
</table>

Employees $\bowtie_{\text{Project}=\text{SecLevel}}$ Projects

<table>
<thead>
<tr>
<th>Employee</th>
<th>Project</th>
<th>SecLevel</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>A</td>
<td>A</td>
<td>Classified</td>
</tr>
<tr>
<td>Black</td>
<td>A</td>
<td>A</td>
<td>Classified</td>
</tr>
<tr>
<td>Black</td>
<td>B</td>
<td>B</td>
<td>Unclassified</td>
</tr>
</tbody>
</table>

- **Inner join**: (most common in SQL) Just returns the rows that match and exists in both table.
- **Outer join**: Returns the same as Inner, but also includes the rows that don’t have a matching value in the second table.
  - **Left outer join**: Yields all of the rows in the first table, including those that do not have a matching value in the second table
  - **Right outer join**: Yields all of the rows in the second table, including those that do not have matching values in the first table
- **Theta join**: Extension of natural join, denoted by adding a theta subscript after the $\bowtie$ symbol

Explain these, then go through several of them with examples.

*I USE SLIDES 03-30 THROUGH 03-36 TO EXPLAIN*

- **Self Joins**
  - Sometimes you need to refer to more than one row (tuple) of the same table (relation) in the same query. Similar to what we did when explaining the rename operator.
  - Example:
    - “Who scored more points than the player with jersey number 10 for any of the regular season games?”
    - To answer this query, we need to compare two tuples p and q of the relation PlayerStats:
      - tuple p, corresponding to the player with jersey number 10, and
      - tuple q, corresponding to the same game as the tuple t, in which u.POINTS > t.POINTS.
    - $S := \rho_X(\text{PlayerStats}) \bowtie_{(X.GameNum=Y.GameNum)} \rho_Y(\text{PlayerStats})$
      $\pi_X(\text{JerseyNum} \land X.points>Y.points \land Y.JerseyNum=10)(S)$

**Aggregate Functions**

*SLIDE 03-37 through 03-38*

some of the functions are `avg`, `sum`, `min`, `max`, `count`

**Algebraic Notes**
● JOIN's are associative:
  
  - $(R \bowtie S) \bowtie T == R \bowtie (S \bowtie T)$

● Technically, “JOIN” chains need no parentheses …:
  
  - $R \bowtie S \bowtie T$
  
  … but it's always better to have clarity rather than showing off.

● JOIN is not commutative unless it is followed by a PROJECTION
  
  - $\Pi L (R \bowtie S) == \Pi L (S \bowtie R)$

● Selection Push-Down
  
  If predicate $\phi$ refers to attributes in $S$ only, then an optimization may be used:
  
  - $\phi(R \bowtie S) == R \bowtie \phi(S)$

1. Lecture notes based on texts by Coronel, Widom, Ullman, and Silberschatz.  ↩