THE PLACE OF USEFUL LEARNING

University of Strathclyde Glasgow

Database Fundamentals – CS990

Database and Web Systems Development - CS952

Image source: https://www.strath.ac.uk/branding/

Course Content

1. **Introduction to Relational Databases** ***(Introduction + Relational Model)***
2. Data Modelling - *(Entity Relationship Modelling + The Enhanced Entity Relationship Model)*
3. Database Design and SQL - *(Logical modelling + Introduction to SQL)*
4. Further SQL - *(Advanced SQL queries + Creating tables with SQL)*
5. Normalisation - *(Normalisation to second normal form + Third normal form)*

The Relational Data Model

- ***Ritchie: Chapter 2, Connolly & Begg: Chapter 3***
- The *relational data model* was first proposed by Edward Codd in a paper written in 1970.
- The relational model has a sound theoretical foundation, which is lacking in some of the other models.
- The objectives of the relational model are:
  - To allow a high degree of data independence. Users' interactions with the database must not be affected by changes to the internal view of data, particularly record orderings and access paths.
  - To provide substantial grounds for dealing with the problems of data semantics, consistency, and redundancy. In particular, Codd's paper introduces the concept of *normalised* relations (details later).

The History of the Relational Data Model

- One of the most significant implementations of the relational model was *System R* which was developed by IBM during the late 1970's.
- System R was intended as a "proof of concept" to show that relational database systems could really be built and work efficiently.
- It gave rise to two major developments:
  - A structured query language called *SQL* which has since become an ISO standard and de facto standard relational language.
  - The production of various commercial relational DBMS products during the 1980s such as *DB2*, *SQL/DS*, and *ORACLE*.
- There are now several hundred relational database systems.
  - Both commercial and open source

Relations

- The relational model is based on the mathematical concept of a *relation*.
  - Since Codd was a mathematician, he used terminology from that field, in particular set theory and logic.
  - We will not deal with these concepts as such, but we need to understand the terminology as it relates to the relational data model.
- A *relation* is represented as a two-dimensional table containing *rows* and *columns* (much like a spreadsheet).
- Relations are used to hold information about the entities to be represented in the database.
  - The *rows* correspond to individual *records*.
  - The *columns* correspond to *attributes* or *fields*.
  - The order of the attributes is unimportant. They can appear in any order and the relation will remain the same.

Basic Terminology - I

- Example: our estate agency may have several branches, and it may wish to store information about its branches. Thus, we have in our data model a *Branch* entity, represented as a relation:

Attribute

Relation name

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

Record

Cardinality (number of rows)

Degree (number of columns)

- As you can see, each column contains the values of a single attribute.
- This may look very much like the file structure that we saw earlier, but note that we are not here talking about storage - all this is about our *data model* only.

Basic Terminology - II

- An *attribute* is a named column of a relation.
  - Correspondingly, a relation comprises one or more named columns representing the attributes of a particular entity type.
- Each attribute has an associated *domain*, i.e. the set of allowable values.
  - A domain is similar to a data type.
  - For example, the domain of the *Street* attribute is the set of all street names in Britain.
- The rows of a relation are called *records*, or more formally, *tuples*.
  - Each record/tuple represents one instance of a particular kind of entity.
  - In the *BRANCH* relation, each record contains six values, one for each attribute, representing the information about a particular branch.
  - The order of the rows of a relation is not important. The rows can appear in a different order and the relation remains the same.

Basic Terminology - III

- The *degree* of a relation is the number of attributes it contains.
  - For example, the *BRANCH* relation has six attributes so it has degree 6.
  - All relations must have at least one attribute, so the degree will always be at least 1.
- The *cardinality* of a relation is the number of records it contains.
  - For example, the *BRANCH* relation has five rows so it has cardinality 5.
  - We may have a relation that has no records, so the cardinality may be 0.
- A *relational database* is a collection of *normalised* relations.
  - We will cover the topic of *normalisation* later…
- Summary of corresponding terms:
  - Relation = Table = File
  - Tuple = Row = Record
  - Attribute = Column = Field

Properties of Relations - I

- The name of a relation is unique.
  - i.e. No two relations may have the same name.
- The name of an attribute is unique only within its relation.
  - so we can have two attributes called *Name* in separate relations, but not in the same relation.
- The values of an attribute are all from the same domain.
  - i.e. we should not allow a postcode to appear in a salary column for example.
- The order of attributes within a relation has no significance.
  - i.e. if we re-order the *columns* of a relation it does not become a different relation.
- The order of rows within a relation has no significance.
  - i.e. if we re-arrange the *rows* of a relation, it does not become a different relation.

Properties of Relations - II

- Each cell of a relation should contain at most *one value*.
  - For example, we cannot store two phone numbers in the same cell.
  - (We shall return to this when we talk about *normal forms* for relations.)
- The records within a relation should all be distinct.
  - i.e. if we examine the values in each row, no two rows should have exactly the same values (no duplicates).
  - Therefore, two rows in a relation should differ in the value of at least one attribute.
  - Note that database systems do not generally enforce this property.
    - **In particular, Microsoft Access and MySQL allow relations to contain duplicate records.**

Keys

- We need to be able to identify uniquely each row in a relation by the values of its attributes.
  - This enables a particular row to be retrieved or related to other records.
- We use *relational keys* for this purpose. These consist of a chosen attribute or set of attributes from the relation in question.
- **Questions:**
  - How do we decide which attribute(s) to choose as the key for a given relation?
  - Is there always a unique key, or do we sometimes have to select one from a number of possibilities?

Candidate Keys

- If a does not contain any superfluous attributes, we say that it is *minimal*. More formally:
  - A key is minimal if removing any attributes would mean that it no longer provides unique identification.
- When choosing a key for a relation, we may pick any minimal key. Minimal keys are therefore called *candidate keys*.
- For example:
  - (*Bno*) is a candidate key for the *BRANCH* relation but (*Bno*, *Postcode*) is not.
- Note that a candidate key may involve more than one attribute
  - (If a key contains more than one attribute we say that it is *composite)*

Properties of Candidate Keys

- A candidate key, *K*, for a relation *R* has the following properties:
- *Uniqueness:*
  - In each row of *R*, the values of *K* uniquely identify that row.
  - In other words: no two rows of *R* can have the same values for *K*.
- *Irreducibility:*
  - No subset of *K* has the uniqueness property.
  - Therefore, *K* cannot consist of fewer attributes.
- The above is simply a formal way of stating what we know about candidate keys.
- Some relations may have several candidate keys.
  - e.g. A *Staff* relation may have *StaffNo* and *NatInsNo* as candidate keys.

Finding Candidate Keys

- An instance of a relation (i.e. a particular table or set of records) cannot be used to *prove* that an attribute or combination of attributes is a candidate key (nor to select an appropriate candidate key).
- In other words, we cannot examine a relation and decide, for example, that *Postcode* is a candidate key simply because no two rows share the same post code.
  - The fact that there are no duplicates at a particular moment in time does not guarantee that duplicates are not possible.
- However, the presence of duplicates may be used to show that a certain attribute or combination of attributes is <u>not</u> a candidate key.
- Therefore, to correctly identify a candidate key, we need to be aware of the *meanings of the attributes in the real world*, and think about whether duplicates *could* arise for a given choice of key.

Primary Keys

- For each relation in the database, we must choose one of its candidate keys to be its *primary key*.
- Since a relation cannot have duplicate rows (by definition), it is always *possible* to uniquely identify each row.
  - This means that in theory every relation has at least one candidate key.
  - Hence, a primary key can always be found.
  - In the worst case, the entire set of attributes could serve as the primary key, but usually some smaller subset is sufficient.
- However, since many database systems allow relations to contain duplicates, this theoretical property does not necessarily apply in practice.
- We are often best served by introducing an *artificial key* attribute if a natural primary key cannot be found. This could be simply a *record number*.

Foreign Keys

- When an attribute appears in more than one relation, its appearance usually represents a *relationship* between records of the relations.
- For example, consider this simple example involving the relationship between *Staff* and *Departments* within the University:

STAFF		
Sno	**Name**	***DeptNo***
SG86	David Hulse	31
SP52	Paul Kingston	49
SJ12	Michael Smith	55
SQ63	Alan Dearle	31

DEPARTMENT		
DeptNo	**Name**	**NumRooms**
31	Computing Science	18
49	Management	15
55	Basket-Weaving	3

- The rows in these relations are linked via the *DeptNo* field.
- When the primary key of one relation appears as an attribute in another relation it is called a *foreign key*.

Relational Database Schemas

- To represent the conceptual schema for a relational database we use the following notation for each relation:

relation_name (*<u>attribute1</u>*, *attribute2*, …, *attributeN*)

- In other words, we write the name of the relation followed by a list of the names of the attributes it contains.
- The primary key attributes are underlined.
- Foreign key attributes should be shown using some distinguishing feature such as a dashed underline.
- We will use this notation where convenient in the future.

Relational Integrity

- The relational data model contains a number of integrity constraints that apply to the relations we create.
- The two principal integrity rules are:
  - *entity integrity rule*
  - *referential integrity rule*
- Both of these rules depend on the concept of *nulls*.
- A *null* represents the value of an attribute that is currently unknown, or is not applicable for a particular record.
- Nulls provide a way to deal with incomplete or exceptional data.
- A null is not the same as a zero value, an empty string, or a string filled with spaces.
  - All of the latter *are* values, whereas a null means *no value*.

Nulls

- Suppose we define a viewing relation as:

  Viewing (*<u>Pno</u>*, *<u>Rno</u>*, *<u>Date</u>*, *Time*, *Comment*)

- The *Comment* field is intended to be filled in after the viewing has taken place.
- Therefore, when the viewing is initially arranged, there is no data to store in the field.
- To represent this initial lack of information, we can use a *null*.
- The use of nulls is a contentious issue:
  - Codd regards the use of nulls as an integral part of the relational model.
  - However, others believe that nulls undermine the solid theoretical foundation of the model.
  - In general, the problem of *missing information* is not yet fully understood and nulls are one way of dealing with the problem, although they may not be the best way.

Entity Integrity

- The entity integrity rule applies to the primary keys of *relations*.

  Entity Integrity Rule:

  In a relation, no attribute of a primary key can be null.

- By definition, a primary key is a *minimal superkey* that is used to identify records uniquely.
  - This means that no subset of the primary is sufficient to provide unique identification.
- If we allow a null for any part of a primary key, we imply that not all of the attributes are needed for unique identification.
  - This contradicts the definition of the primary key.
- For example, in the viewing relation, the primary key is (*Pno*, *Rno*, *Date*). We should not be able to insert a new row for which *Pno*, *Rno*, or *Date* are *null*.

Referential Integrity

- The second integrity rule applies to *foreign keys*.

  Referential Integrity Rule:

  If a relation contains a foreign key, either the foreign key value must match the value of a candidate key of a record in the home relation, or the foreign key value must be wholly null.

- For example, in the *Staff* and *Department* example, the *Staff* relation contains a foreign key from the *Department* relation.
  - i.e. The *DeptNo* field that relates a staff member to the department in which they work.
- The rule states that we should not be able to insert a staff member record for which the *DeptNo* field refers to a department that does not exist in the *Department* relation.
- We can set the *DeptNo* field to *null*, however, in the *Staff* relation.

Course Content

1. **Introduction to Relational Databases** ***(Introduction + Relational Model)***
2. Data Modelling - *(Entity Relationship Modelling + The Enhanced Entity Relationship Model)*
3. Database Design and SQL - *(Logical modelling + Introduction to SQL)*
4. Further SQL - *(Advanced SQL queries + Creating tables with SQL)*
5. Normalisation - *(Normalisation to second normal form + Third normal form)*

Questions

Database Fundamentals – CS990

Database and Web Systems Development - CS952

Image source: https://www.strath.ac.uk/branding/