Relational Databases
Overview

Full Description of the Relational Model
- Mapping ER diagrams to the Relational Model
- Integrity Constraints and the Relational Model

Querying A Relational Database
- The Relational Algebra
- Structured Query Language
- Updates in the Relational Model

More on Design
- Normalisation

Miscellaneous
- Survey of Relational DBMS
- Oracle Tools

The Relational Model

This was introduced by Codd in 1970 and provides:
- a simple data structure for modelling all data
- which is mathematically based
- and is the current standard for implementation data models
  - more than 70% of new database licenses are relational

Consequences :
- Simplicity means that correctness is easier to establish
- Standardisation means that distributed data can be combined more easily
- It also means that improvements to the facilities and the implementation can be shared more easily

Description of the Relational Model

All of the information stored in a Relational Database is held in Relations.

There are **no other data structures** which hold information!

A relation may be thought of as a table, e.g.

STUDENT

name	**matric**	**exam1**	**exam2**
Gedge	0301023	12	58
Kerr	0702361	66	90
Fraser	0700123	50	65

A relation has
- a **name**
- an unchanging set of **columns** which are named and typed
- a time varying set of **rows**

Reviewing Definitions

A **relation** is a rectangular data structure in which the columns are named and typed and the rows are left unidentified
- A column is called an **attribute** or a **field**
- A **domain** is the set of atomic values which can appear in a column
- A **tuple** or **record** is a row of a relation
- The **degree** of a relation is the number of attributes and the **cardinality** is the number of tuples

A **relational schema** defines a relation:

e.g. Student (Text names, Number matric, Number ex1, Number ex2)
- A **relational database schema** defines all the relations in a database
- A **relational database instance** is a set of relations realizing a relational database schema

Any set of attributes which are unique for each row are called **Keys**, or **Candidate Keys** and one such set is chosen to identify the tuples and is called the **Primary Key** - it is underlined, e.g. *matric* in the above

Characteristics of the Relational Model

Key Slide

1. **There are no duplicate tuples** - since the tuples are a set
   - This implies that a primary key always exists (at worst all of the attributes)
2. **The tuples are unordered** - again a property of sets
3. **The attributes are also unordered**
4. **All values are atomic**
   - No cell contains multiple values
     - e.g. one number not a set of numbers
     - not a subsidiary record either
   - This is called **First Normal Form** and is the start point for Normalisation
5. **Unknown Values Must be Represented**
   - These are called **null** values

Connecting Data in a Relational DB

Key Slide

There are **only two ways to do this**

1. Two pieces of data are related if they are in the **same record**

   ```
   Student( "John", 9901234, 72, 67 )
   ```

   means that the student with matriculation number 9901234 is called John and has marks 72 and 67.

2. The values in two records are related if the records are connected by a **foreign key**

   Given another column in Student with the supervisor's staff number in it and a staff relation:

   ```
   Staff( name, staffNumber, department )
   ```

   then:

   ```
   Student( "John", 9901234, 72, 67, 6543 )

   Staff( "June", 6543, 55 )
   ```

   i.e. the student with matriculation number 9901234 has a supervisor called "June" and she works for the department whose key is 55

Views

Key Slide

The data is stored in **base tables** which have the relational structure, but, it is useful to maintain other relations which are calculated from the base tables - **views** (called **stored queries** in Access**)**

- e.g. for security - access can be restricted to part of a table

Views (and queries in general) usually consist of a mixture of two kinds of operation:

- **restricting** one table - some columns and some rows

  ProfessorNames = the names of *Staff* where title = "Prof"

- **joining** two columns using a foreign key

  StudentSupervisor = all columns of *Staff* and *Student* where the *supervisor* column of *Student* = the *staffNumber* column of *Staff*

- or using a **combination** of several of these
  - joining student, supervisor and department and then showing student and department names

Note – This only works because the results of queries are always relations!

Mapping an ER Model into the R.M.

We can now prescribe a step-by-step methodology, although for individual applications we may depart from this

The ER diagram for the Company database is used as an example

The tables which we will create are shown on the next slide

Do not include derived attributes, since they can be calculated accurately whenever you need them

- (e.g. age from dateOfBirth, also counts, sums, etc....)

Overview

- Create tables from Strong Entities, then Weak Entities
- Add foreign keys into the tables from 1 : 1, 1 : N relationships
- Create new tables from M:N relationships
- Create new tables from multi-valued attributes
- Consider non-binary relationships

Employee Tables (From the Company ER diagram)

Employee *house & street omitted – lack of space, fName & lName are combined (unrealistic)*

ni#	name	city	salary	gender	bDate	*dept*	*supervisr*
123	Jane Brown	Glasgow	12000	F	12/06/76	5	122
122	John Smith	Stirling	23000	M	14/05/55	4	null
124	Lorna Green	Stirling	19000	F	01/01/67	4	122

Department

dNum	dName	*mgrNi#*	mgrStartDate
5	Admin	123	12/02/1999
4	R & D	124	14/03/2001
6	Production	null	null

Project

pNum	pName	pLocation	*conDept*
12	Jupiter	Glasgow	4
14	Mars	London	4
15	Uranus	Glasgow	6

Dependent

empni#	dpdName	dpdBDate	dpdGender	relship
123	Keith	01/04/2000	M	Child
124	Mary	01/12/1905	F	Mother

WorksOn

emp	*project*	hours
123	12	5
123	14	20
122	14	10

What We DON'T Want

In the Employee table below, we can see all details about an employee including which department the employee works for – name and number

This is an **unecessary duplication** of department name and number.

- If the department changed name, it would have to be altered in all these records – **wasted work**
- The department name could be entered differently for the same department number, leading to a **lack of consistency**

NI#	NAME	Address Etc	DeptNum	DeptName
423	Joe	…..	5	Research
122	John	…..	6	Admin
144	Jean	…..	4	Personnel
444	Jane	…..	6	Admin
321	Jack	…..	6	Admin

Keys

Primary Keys

- uniquely identify each tuple
- are shown underlined
- can be **composite** (consist of more than one attribute)
- but try to keep them **simple** and **small** (in terms of number of bits)
  - it would be more efficient to choose an integer rather than text
  - you could use an automatically generated sequence number if the alternative is very complex
    - an address, which is long and can easily be mistyped and consists of several attributes (this can make queries very cumbersome)

Foreign Keys

- columns in one table which refer to the primary key of another table
- they show the relationship between the tables
- they are used subsequently to connect the data (e.g. find the name of the projects that are controlled by department 5)

Choice of Column Names

You do not have to keep the exact name that is in the ER diagram, although if you do, it makes comparing the two simpler.

- Choose a meaningful name (as with java variables)

Column names do not have to be unique within your database, but they must be unique within the relation.

If you have two identically named columns in different relations, you may need to use the name of the relation to refer to them (in SQL etc)
e.g. Department.name and **Project.name**

But I would name a foreign key column with the Entity or Relationship name not the primary key attribute name e.g.

- department not dNum in Employee

I am (mostly) using upper case to start table names, and lowercase to start column names. This is similar to classes and instance variables in java.

1. Strong Entity Types

For each strong entity type, we create a relation with:

- one column for each simple attribute
- one column for each part of a composite attribute (e.g. address)
- the **Primary Key** is chosen to be one of the **candidate key** attributes
  - It's probably better to put all components of the primary key together at the start of the list of attributes, however not all implementations insist on this
- leave out multi-valued attributes (they are considered later)

Employee (ni#, fName, lName, bDate, house, street, city, gender, salary, ...)

Project (pNum, pName, pLocation, ...)

Department(dNum, dName, ...)

The dots mean that we are possibly going to add more columns to these tables later

2. Weak Entity Types

As for strong entities, but add the Primary Key of the 'owner' entity

- as the Partial Key of the weak entity is only unique among the owner entities of the same key

For instance, each dependent is identified by knowing their name AND ALSO which Employee they are dependent on. *(We are assuming here that dependents of one employee do not have the same name....)*

Dependent(*empni#*, dpdName, dpdBDate, dpdGender, relship)

These added column(s) are **Foreign Keys** which refer to the primary key of the 'owner' entity **AND** they are part of the primary key of this relation

The **composite** Primary Key, which uniquely identifies each tuple, is composed of

- the Partial Key of the weak entity
- the Primary Key of the 'owner' entity

3. Binary 1-N Relationships

Relationships are mapped into foreign keys. 1-N relationships become a column in one relation referring to a primary key in the other.

But which way? Example **Employee works for Department:**

Loss of Primary Key Constraint

Department

Dnumb	Dname	Emp
5	R&D	1111
5	R&D	1112
6	Admin	1113
6	Admin	1114

Employee

Staff#	Name	Age	Salary
1111	J Black	25	£14k
1112	J Brown	29	£18k
1113	J Green	33	£43k
1114	J White	18	£32k

This is ok

or

Department

Dnumb	Dname
5	R&D
6	Admin

Employee

Staff#	Name	Age	Salary	Dept
1111	J Black	25	£14k	5
1112	J Brown	29	£18k	5
1113	J Green	33	£43k	6
1114	J White	18	£32k	6

Binary 1-N Relationships contd

Represent this relationship by adding column(s) to the relation on the N-side containing

- the primary key from the relation on the 1-side
- any attributes of the relationship – *these always go next to the foreign key*

Each department is worked for by many employees, but each employee works for only one department

Employee (ni#, ...salary, *dept*)

dept is a foreign key referencing **dNum** in the Department table

Each department controls many projects, but each project is controlled by only one department

Project (pNum, pName, pLocation, *conDept*)

conDept is a foreign key referencing **dNum** in the Department table

Binary 1-N Relationships contd

Each employee can supervise many employees, but each employee can only have one supervisor

Employee (ni#, fName, lName, bDate, house, street, city, salary, gender, *dept, super*)

super is a foreign key referencing **ni#** in the Employee table

- *This is a recursive relationship so the foreign key refers to other tuples in the same table*

Each Employee can have many Dependents

Dependent(*empni#*, dpdName, dpdDateOfBirth, dpdGender, relship)

This relationship has already been considered because the Dependent is a weak entity. No further action need be taken.

empni# is a foreign key referencing **ni#** in the Employee table

4. Binary 1 - 1 Relationships

This is more tricky – we must also use participation constraints

Example - **Employee manages Department**

The second one introduces null values which wastes space - put foreign key in the total participation side

Department

Dnumb	Dname	Mgr
5	R&D	1111
6	Admin	1113

Employee

Staff#	Name	Age	Salary
1111	J Black	25	£14k
1112	J Brown	29	£18k
1113	J Green	33	£43k
1114	J White	18	£32k

If both are total, either side will do or even make one table

If both are partial, try to minimise null values

Department

Dnumb	Dname
5	R&D
6	Admin

Employee

Staff#	Name	Age	Salary	Manages
1111	J Black	25	£14k	5
1112	J Brown	29	£18k	null
1113	J Green	33	£43k	6
1114	J White	18	£32k	null

4. Binary 1 - 1 Relationships

Represent the relationship by adding a column(s) to **one** of the relations of the participating entity types

- If one of the entities has total participation, add the column to that side to avoid nulls
- If both sides are total,
  - the foreign key can go either side
  - or the two tables could be merged
- If both sides are partial, there needs to be judgement on which side would create the fewer null values
  - e.g. if some departments had no managers, we would still put the foreign key in Department, since there are more employees who are not managers than *vice versa*

The contents of this column are **Foreign Keys** which match the primary key(s) of the other relation.

Attributes of the relationship are also appended next to the FK column

5. Binary M-N Relationships

Now the simple addition of a foreign key column to one table breaks down - whichever side we add the column to there will be duplication

We must add a **new table** pairing the primary keys of both:

Project

Pnumb	Pname
22	Mars
23	Galaxy
24	Snickers

WorksOn

Proj	Emp
22	1111
22	1113
22	1114
23	1111
23	1112
23	1114
24	1113
24	1114

Employee

Staff#	Name	Age	Salary
1111	J Black	25	£14k
1112	J Brown	29	£18k
1113	J Green	33	£43k
1114	J White	18	£32k

J White works on Snickers

Binary M-N Relationships contd.

Create a new relation to hold this, containing Foreign Key columns pointing to participating entities and further columns for attributes of the relationship
- i.e. the attributes of the relationship must also go in this table

Examples Employee works on Project
- Each Employee can work on many projects
- Each project can be worked on by many employees

WorksOn (*emp, project*, hours)

- **emp** is a foreign key referencing **ni#** in the Employee table
- **project** is a foreign key referencing **pNum** in the Project table

The Primary Key of this relation will be **composite** because it consists of the pair of foreign keys (and of course the foreign keys themselves may be composite....).
- If there are no attributes of the relationship, the relation will consist solely of the primary key attributes

6. Non-Binary Relationships

Create a new relation with columns for the Primary Keys of each participating entity relation. Add further columns to represent the attributes of the relationship.
- There are no non-binary relationships in the company database.

E.g. Inter - Library - Loan Linking Borrower, Book & Owing Library:

ILL (*borrower, bookISBN, ownerID*, dateDue)

Careful consideration is needed here to determine the Primary Key of this relation.
- It could consist of all three primary keys.
- The owning library has not been included since a borrower is not expected to borrow the same book at the same time from different libraries.
- The dateDue, an attribute of the relationship, is included since the same borrower may borrow the same book more than once.

7. Multi-Valued Attributes

Create a new relation holding one column(s) for the Primary Key of the relation representing the entity and another for the values.
The locations of the Department are multi-valued i.e. there could be more than one location for each department
- The primary key of this relation will be composite. Both the department number and the location is needed to make each tuple unique

DeptLocs(*dLdNum*, dLocation)

NB If any attributes of an M:N relationship are multi-valued (e.g. When someone worked on a project, which could be more than once) then these should also become part of the primary key.

FINALLY

Use these seven steps to transform your ER diagram into tables.

Then consider the tables – do they make sense? If not, maybe your ER diagram is wrong!

ALSO, WE MAY DEPART FROM THIS METHODOLOGY IF WE WANT TO

e.g.
- split an entity into two relations – commonly referenced attributes and rarely referenced ones (especially if the latter are large)
- represent some 1 : N relationships by relations if participation is very partial – i.e. there are only a few linked records
  - e.g. there could be a table of disability records for students with a foreign key from the main student table to the disability record
  - but most students don't have these, so the foreign key column would mostly be null values
  - therefore maybe create a separate table pairing matriculation number with disability record number

Integrity Constraints on Relational DB's

The meaningfulness of the data in the database is mostly conveyed through the structure of the tables and the names used.

However, some of the meaning is captured in the form of integrity constraints.

These limit the values which can be inserted into the database structure.

There are two main categories of constraint:

- **Inherent Integrity Constraints** form a set of rules which must hold for all relational DB's and are enforced by the DBMS
- **Enterprise Constraints** are those rules which are specific to the particular application
  - These might be expressed in the query language – see next few slides
  - Or they might need to be expressed in the application program if they are complex

Three Inherent Integrity Constraints

The first two constraints are specified on individual relations, and are supported by most DBMS:

1. Key Constraints

- Reminder : **Candidate key** of a relation/table - any set of attributes of that table that uniquely identify that tuple/row in the table. One candidate key is chosen to be the primary key.
- e.g. an Employee table : Staff Number and National Insurance Number are both candidate keys. Choose Staff Number to be the primary key.
- **Key Constraint** : a candidate key value for each table MUST be unique for every row in that table.

Name for feedback –i.e. what went wrong

In Oracle: *Primary keys in Oracle are automatically unique*

```
staffNum  VARCHAR2(6)   CONSTRAINT pk_Emp PRIMARY KEY

niNumber VARCHAR2(10) CONSTRAINT un_NI  UNIQUE
```


Integrity Constraints continued

2. Entity Integrity

- The primary key of a table must be non – null.
  - (Because this is what is used to identify each row)
- In Oracle – Primary keys are automatically non-null

3. Referential Integrity (Foreign keys)

- This constraint is specified between two relations:
- Any attributes of a relation which are foreign keys to another relation must take values which either
  - exist in the second relation
  - or are null

(see next slide)

Example of Foreign Keys

Courses taken by a lecturer MUST be in the Course table

- Some lecturers (RI) may not take courses in which case the course entry in the *Lecturer* table will be null

Course

number	name
229	DBIS
220	PSD
222	AP

Lecturer

name	takes
RC	229
RW	220
RS	231
RI	**null**

?

- *takes* in the *Lecturer* table is a foreign key referencing number in the Course table

```
takes   INTEGER   CONSTRAINT fk_TAKES takes REFERENCES
   Course(Number)
```

The reference Takes 231 from *Lecturer* to *Course* refers to something which doesn't exist (this is called a **dangling pointer**)

- and this implies incompleteness and the possibility of queries failing to be completed. e.g. tell me the name of courses that “RS” teaches.

Enterprise Constraints

In addition to the inherent constraints, it is useful to specify other constraints which ensure that the data held in the database remains sensible

Again there are a number of constraint kinds which might be specified:

- **non-null constraints** applied to non-primary-key fields
  - e.g. Account table – type, balance and branch must be supplied
  - You can specify total participation by constraining the foreign keys to be non-null
- **check constraints** are general constraints which limit the value
  - the values in a column must lie in some range
  - the values in one column must be less than those in another column of the same record

See Examples in CreateAll Tables.sql and also online (Oracle SQLReference)

```
CONSTRAINT      ck_cust_sex     CHECK  (sex in ('M', 'F')),

CONSTRAINT      ck_cust_id      CHECK  (id between 100 and 999)
```


Enforcing Integrity Constraints

The DBMS must continually check that constraints are not violated every time an update occurs

Three options for how to proceed:

- allow the update anyway – essentially ignoring the constraint
- refuse to perform violating update
- compensate in some way

When using a DBMS, always discover what it does!

- early versions of mySQL ignore foreign keys,
- Access allows cascading for updates and deletes, ...

Compensation is performed by:

- **Cascading** - make a compensating change to the current tuple then check everything that refers to it
  E.g. Delete a branch – delete all accounts in that branch (?!)
- **Restricting** - only change tuples which don't violate constraints
- **Nullifying** - set Foreign Keys to null if referential integrity is violated
- Setting the value to a **default**

	on delete	**on update (insert and modify)**
Cascade	delete this record as well	change the foreign key value in this table as well
Set null	set the foreign key value in this table to null	set the foreign key value in this table to null
Set default	set the foreign key value in this table to a default value	set the foreign key value in this table to a default value
No action	do nothing	do nothing

ON DELETE CASCADE, ON DELETE SET NULL are the only options permitted in Oracle.

In the Account Table :

```
CONSTRAINT fk_Abranch FOREIGN KEY (inBranch)
    REFERENCES Branch (branchNo) ON DELETE CASCADE
```

But is this sensible? Always consider the situation.

Examples of Preferred Action on Updates

An employee leaves the company and his details are deleted. What should happen to details about

- his dependents
- the projects he worked on
- the employees that he supervised
- the department that he managed

We discover that an employee has the wrong NI#. What should happen to all the foreign keys in related tables?

A department closes. What should happen to all the foreign keys in related tables?