X SEMINAR ON SOFTWARE FOR NUCLEAR, SUBNUCLEAR AND APPLIED PHYSICS
2 – 8 JUNE 2013, PORTO CONTE, ALGHERO

Databases in experimental Physics: theory and practice

L. Tomassetti

University of Ferrara and INFN

Notice!

- I’m teaching the course
  “Databases and Laboratory”
  to Computer Science Bachelor's students

  and “Advanced Databases”
  to Computer Science Master’s students

Summary

- Introduction
  - Why Databases are important for Physicists
- A 1-semester database course in half-an-hour !!!
- Some applications/use cases/examples:
  - Good design (?)
  - Problems arising from database design issues

Introduction

- Database System are widely used in several fields
- We are interested in how they can make our work easier:
  - Administrative tasks
  - Organizational tasks
  - Data store and management
  - **Metadata** store and management
  - **Backend** for services in use by experiments

Introduction

- Please, let me ask you a couple of questions:
  - How many of you are working in HEP experiments?
  - How many of you have ever used the Grid Infrastructure (directly)?
  - ... ever used the Grid Infrastructure through a ‘custom’ service?
  - ... ever used (a query, for example) a database management system (directly)?
  - ... ever used a Wiki, Tracking software, ...?
- No, I’ll not ask you:
  - ... ever used Amazon/eBay/AppStores/...?
  - ... ever used Facebook/Twitter/...?

Introduction

- Just a few examples to start with:
  - Publications databases (WoS, Scopus, INFN, ...)
  - Organizational databases (experiments, ...)
  - Bookkeeping databases
  - Conditions databases
  - Grid access services (Dirac, ...)

Introduction

- As ‘simple’ users we may consider services as black-boxes and ignore the internals
- As soon as we need something more (in Physics applications of course) it **can be useful** to know some details...
- As soon our supervisors ask for writing analysis/ core software/utility code we **have to** know (almost) all details...

Introduction

- Database are relatively (maybe too) simple, we just need to understand a few concepts and apply them correctly
- We do not need to be database experts (unless we have to develop core software tightly connected to dbms)
- I'll limit the discussion to Relational Databases

DATABASE IN THEORY

Content

- Introduction & Conceptual Modeling
- Database System: Concepts and Architecture
- Entity–Relationship Model
- Relational Model
- E–R to Relational Model Mapping
- Relational Algebra
- SQL

You can decide:
(a) my speed
(b) which parts we should discuss

I would suggest at least:
- Relational Model + SQL
- E-R + Algebra

Introduction & Conceptual Modeling

Basic Definitions

- **Database**: A collection of related data.
- **Data**: Known facts that can be recorded and have an implicit meaning.
- **Mini-world**: Some part of the real world about which data is stored in a database. For example, student grades and transcripts at a university.
- **Database Management System** (DBMS): A software package/ system to facilitate the creation and maintenance of a computerized database.
- **Database System**: The DBMS software together with the data itself. Sometimes, the applications are also included.

Typical DBMS Functionality

- **Define** a database: in terms of data types, structures and constraints
- **Construct** or **Load** the Database on a secondary storage medium
- **Manipulating** the database: querying, generating reports, insertions, deletions and modifications to its content
- **Concurrent Processing** and **Sharing** by a set of users and programs – yet, keeping all data valid and consistent

Typical DBMS Functionality

- Other features:
  - Protection or Security measures to prevent unauthorized access
  - “Active” processing to take internal actions on data
  - Presentation and Visualization of data

Main Characteristics of the Database Approach

- Self-describing nature of a database system: A DBMS catalog stores the description of the database (the description is called meta-data). This allows the DBMS software to work with different databases.

- Insulation between programs and data: Called program-data independence. Allows changing data storage structures and operations without having to change the DBMS access programs.

Main Characteristics of the Database Approach

- Data Abstraction: A data model is used to hide storage details and present the users with a conceptual view of the database.
- Support of multiple views of the data: Each user may see a different view of the database, which describes only the data of interest to that user.

Main Characteristics of the Database Approach

- Sharing of data and multiuser transaction processing : allowing a set of concurrent users to retrieve and to update the database. Concurrency control within the DBMS guarantees that each transaction is correctly executed or completely aborted. OLTP (Online Transaction Processing) is a major part of database applications.

Advantages of Using the Database Approach

- Controlling redundancy in data storage and in development and maintenence efforts.
- Sharing of data among multiple users.
- Restricting unauthorized access to data.
- Providing persistent storage for program Objects
- Providing Storage Structures for efficient Query Processing

Advantages of Using the Database Approach

- Providing backup and recovery services.
- Providing multiple interfaces to different classes of users.
- Representing complex relationships among data.
- Enforcing integrity constraints on the database.
- Drawing Inferences and Actions using rules

Additional Implications of Using the Database Approach

- Potential for enforcing standards: this is very crucial for the success of database applications in large organizations Standards refer to data item names, display formats, screens, report structures, meta-data (description of data) etc.

- Reduced application development time: incremental time to add each new application is reduced.

Additional Implications of Using the Database Approach

- Flexibility to change data structures: database structure may evolve as new requirements are defined.
- Availability of up-to-date information – very important for on-line transaction systems such as airline, hotel, car reservations.
- Economies of scale: by consolidating data and applications across departments wasteful overlap of resources and personnel can be avoided.

When not to use a DBMS

- When no DBMS may suffice:
  - If the database system is not able to handle the complexity of data because of modeling limitations
  - If the database users need special operations not supported by the DBMS.

Database System: Concepts and Architecture

Data Models

- **Data Model**: A set of concepts to describe the structure of a database, and certain constraints that the database should obey.

- **Data Model Operations**: Operations for specifying database retrievals and updates by referring to the concepts of the data model. Operations on the data model may include basic operations and user-defined operations.

Categories of data models

- **Conceptual** (*high-level*, semantic) data models: Provide concepts that are close to the way many users perceive data. (Also called entity-based or object-based data models.)

- **Physical** (*low-level*, internal) data models: Provide concepts that describe details of how data is stored in the computer.

- **Implementation** (representational) data models: Provide concepts that fall between the above two, balancing user views with some computer storage details.

History of Data Models

- **Relational** Model: proposed in 1970 by E.F. Codd (IBM), first commercial system in 1981-82. Now in several commercial products (DB2, ORACLE, SQL Server, SYBASE, INFORMIX) and several open-source products (MySQL, PostgreSQL, ...).

- **Network** Model: the first one to be implemented by Honeywell in 1964-65 (IDS System). Adopted heavily due to the support by CODASYL (CODASYL - DBTG report of 1971). Later implemented in a large variety of systems - IDMS (Cullinet - now CA), DMS 1100 (Unisys), IMAGE (H.P.), VAX -DBMS (Digital Equipment Corp.).

- **Hierarchical** Data Model: implemented in a joint effort by IBM and North American Rockwell around 1965. Resulted in the IMS family of systems. The most popular model. Other system based on this model: System 2k (SAS inc.)

History of Data Models

- **Object-oriented** Data Model(s): several models have been proposed for implementing in a database system. One set comprises models of persistent O-O Programming Languages such as C++ (e.g., in OBJECTSTORE or VERSANT), and Smalltalk (e.g., in GEMSTONE). Additionally, systems like O2, ORION (at MCC - then ITASCA), IRIS (at H.P.- used in Open OODB).

- **Object-Relational** Models: Most Recent Trend. Started with Informix Universal Server. Exemplified in the latest versions of Oracle-10i, DB2, and SQL Server etc. systems.

Schemas versus Instances

- **Database Schema**: The description of a database. Includes descriptions of the database structure and the constraints that should hold on the database.
- **Schema Diagram**: A diagrammatic display of (some aspects of) a database schema.
- **Schema Construct**: A component of the schema or an object within the schema, e.g., STUDENT, COURSE.
- **Database Instance**: The actual data stored in a database at a particular moment in time. Also called database state (or occurrence).

Database Schema Vs. Database State

- **Database State**: Refers to the content of a database at a moment in time.
- **Initial Database State**: Refers to the database when it is loaded
- **Valid State**: A state that satisfies the structure and constraints of the database.
- Distinction:
  - The database schema changes very infrequently. The database state changes every time the database is updated.
  - Schema is also called intension, whereas state is called extension.

Three-Schema Architecture

- Proposed to support DBMS characteristics of:
  - Program-data independence.
  - Support of multiple views of the data.

Three-Schema Architecture

- Defines DBMS schemas at three levels:
  - Internal schema at the internal level to describe physical storage structures and access paths. Typically uses a physical data model.
  - Conceptual schema at the conceptual level to describe the structure and constraints for the whole database for a community of users. Uses a conceptual or an implementation data model.
  - External schemas at the external level to describe the various user views. Usually uses the same data model as the conceptual level.

Three-Schema Architecture

- Mappings among schema levels are needed to transform requests and data. Programs refer to an external schema, and are mapped by the DBMS to the internal schema for execution.

Data Independence

- **Logical** Data Independence: The capacity to change the conceptual schema without having to change the external schemas and their application programs.

- **Physical** Data Independence: The capacity to change the internal schema without having to change the conceptual schema.

Data Independence

- When a schema at a lower level is changed, only the mappings between this schema and higher-level schemas need to be changed in a DBMS that fully supports data independence. The higher-level schemas themselves are unchanged. Hence, the application programs need not be changed since they refer to the external schemas.

DBMS Languages

- **Data Definition Language** (DDL): Used by the DBA and database designers to specify the conceptual schema of a database. In many DBMSs, the DDL is also used to define internal and external schemas (views). In some DBMSs, separate storage definition language (SDL) and view definition language (VDL) are used to define internal and external schemas.

DBMS Languages

- **Data Manipulation Language** (DML): Used to specify database retrievals and updates.
  - DML commands (data sublanguage) can be embedded in a general-purpose programming language (host language), such as COBOL, C or an Assembly Language.
  - Alternatively, stand-alone DML commands can be applied directly (query language).

DBMS Languages

- High Level or Non-procedural Languages: e.g., SQL, are set-oriented and specify what data to retrieve than how to retrieve. Also called declarative languages.
- Low Level or Procedural Languages: record-at-a-time; they specify how to retrieve data and include constructs such as looping.

Entity – Relationship Model

ER Model

- Example Database Application (COMPANY)
- ER Model Concepts
  - Entities and Attributes
  - Entity Types, Value Sets, and Key Attributes
  - Relationships and Relationship Types
  - Weak Entity Types
  - Roles and Attributes in Relationship Types
- ER Diagrams - Notation
- ER Diagram for COMPANY Schema

Example COMPANY Database

- Requirements of the Company (oversimplified for illustrative purposes)
  - The company is organized into DEPARTMENTs. Each department has a name, number and an employee who manages the department. We keep track of the start date of the department manager.
  - Each department controls a number of PROJECTs. Each project has a name, number and is located at a single location.
  - We store each EMPLOYEE's social security number, address, salary, sex, and birthdate. Each employee works for one department but may work on several projects. We keep track of the number of hours per week that an employee currently works on each project. We also keep track of the direct supervisor of each employee.
  - Each employee may have a number of DEPENDENTs. For each dependent, we keep track of their name, sex, birthdate, and relationship to employee.

ER Model Concepts

- Entities and Attributes
  - Entities are specific objects or things in the mini-world that are represented in the database. For example the EMPLOYEE John Smith, the Research DEPARTMENT, the ProductX PROJECT
  - Attributes are properties used to describe an entity. For example an EMPLOYEE entity may have a Name, SSN, Address, Sex, BirthDate
  - A specific entity will have a value for each of its attributes. For example a specific employee entity may have Name='John Smith', SSN='123456789', Address ='731, Fondren, Houston, TX', Sex='M', BirthDate='09-JAN-55‘
  - Each attribute has a value set (or data type) associated with it – e.g. integer, string, subrange, enumerated type, …

Types of Attributes (1)

- Simple
  - Each entity has a single atomic value for the attribute. For example, SSN or Sex.
- Composite
  - The attribute may be composed of several components. For example, Address (Apt#, House#, Street, City, State, ZipCode, Country) or Name (FirstName, MiddleName, LastName). Composition may form a hierarchy where some components are themselves composite.
- Multi-valued
  - An entity may have multiple values for that attribute. For example, Color of a CAR or PreviousDegrees of a STUDENT. Denoted as {Color} or {PreviousDegrees}.

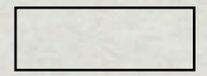
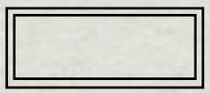
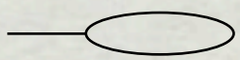
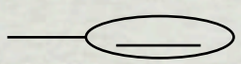
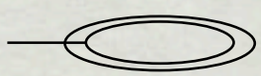
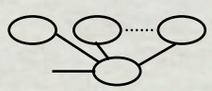
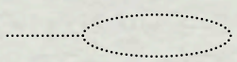
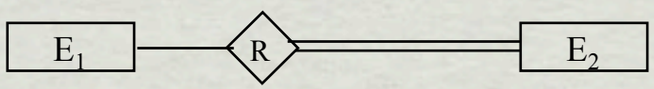
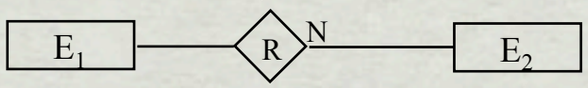
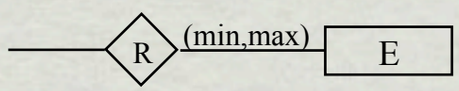
Types of Attributes (2)

- In general, composite and multi-valued attributes may be nested arbitrarily to any number of levels although this is rare. For example, PreviousDegrees of a STUDENT is a composite multi-valued attribute denoted by {PreviousDegrees (College, Year, Degree, Field)}.

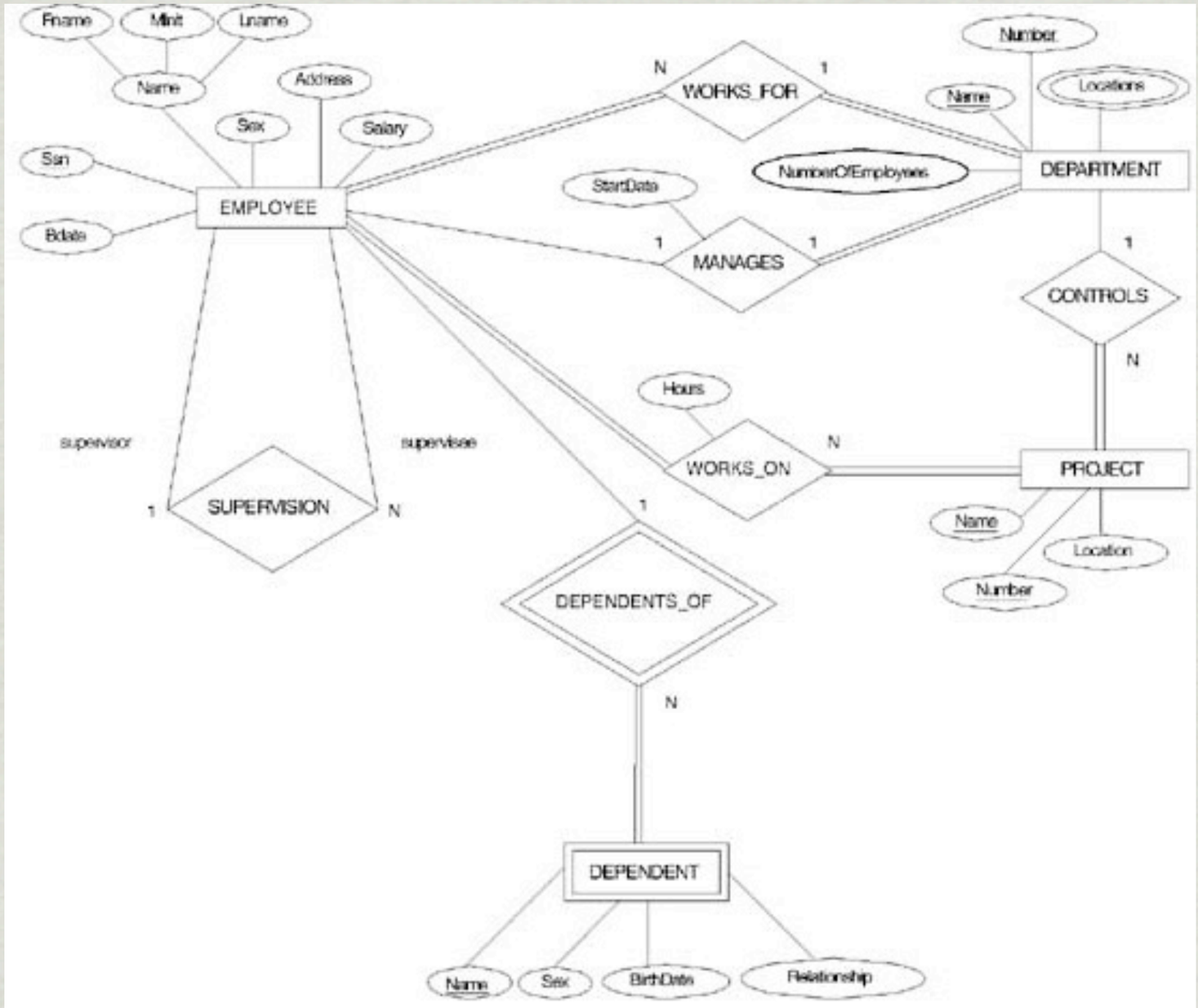
Entity Types and Key Attributes

- Entities with the same basic attributes are grouped or typed into an entity type. For example, the EMPLOYEE entity type or the PROJECT entity type.
- An attribute of an entity type for which each entity must have a unique value is called a key attribute of the entity type. For example, SSN of EMPLOYEE.
- A key attribute may be composite. For example, VehicleTagNumber is a key of the CAR entity type with components (Number, State).
- An entity type may have more than one key. For example, the CAR entity type may have two keys:
  - VehicleIdentificationNumber (popularly called VIN) and
  - VehicleTagNumber (Number, State), also known as license_plate number.

SUMMARY OF ER-DIAGRAM NOTATION FOR ER SCHEMAS

Symbol	Meaning
	ENTITY TYPE
	WEAK ENTITY TYPE
	RELATIONSHIP TYPE
	IDENTIFYING RELATIONSHIP TYPE
	ATTRIBUTE
	KEY ATTRIBUTE
	MULTIVALUED ATTRIBUTE
	COMPOSITE ATTRIBUTE
	DERIVED ATTRIBUTE
	TOTAL PARTICIPATION OF E_2 IN R
	CARDINALITY RATIO 1:N FOR E_1:E_2 IN R
	STRUCTURAL CONSTRAINT (min, max) ON PARTICIPATION OF E IN R

ER DIAGRAM – Entity Types are: EMPLOYEE, DEPARTMENT, PROJECT, DEPENDENT



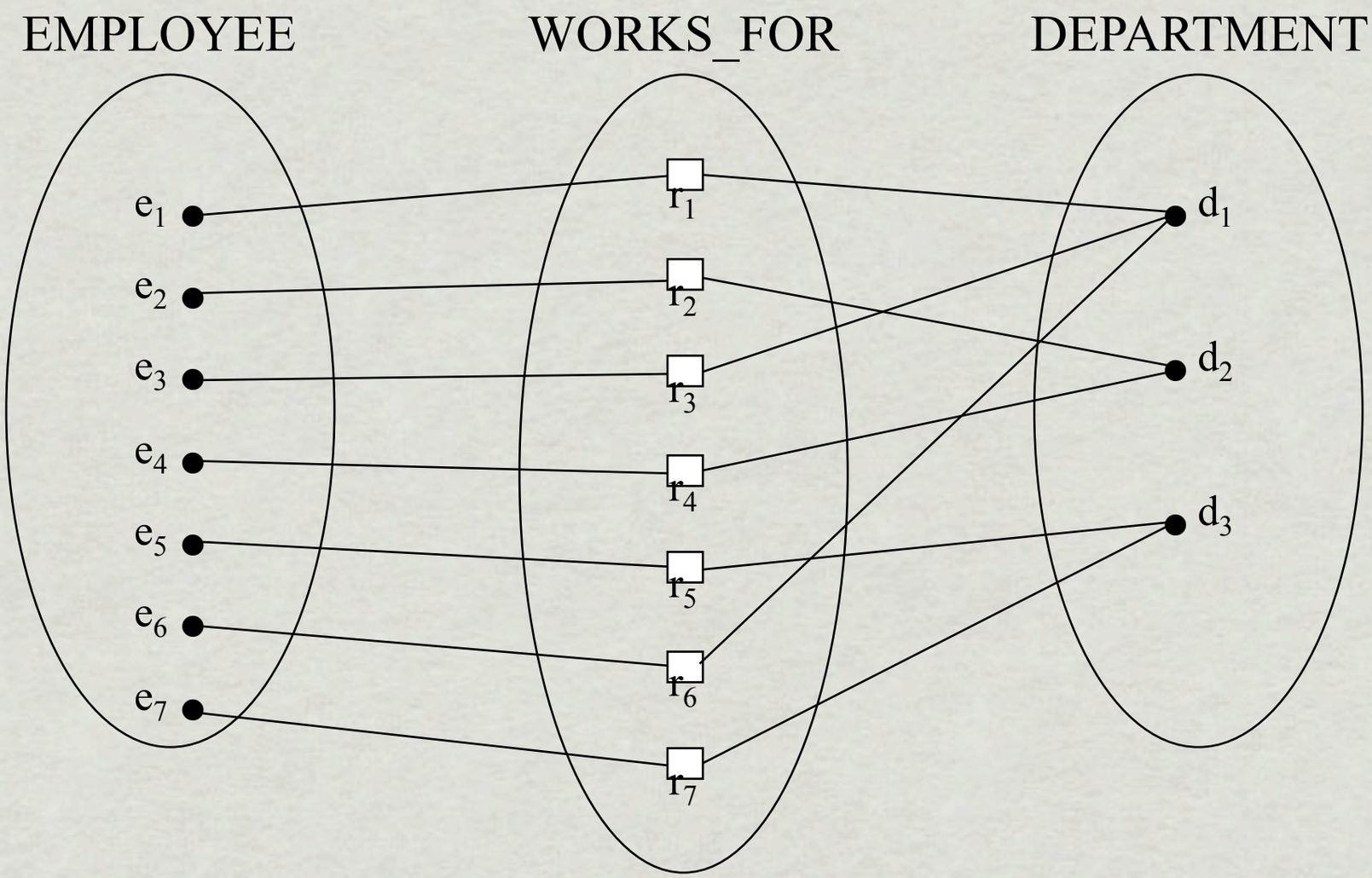
Relationships and Relationship Types (1)

- A relationship relates two or more distinct entities with a specific meaning. For example, EMPLOYEE John Smith works on the ProductX PROJECT or EMPLOYEE Franklin Wong manages the Research DEPARTMENT.

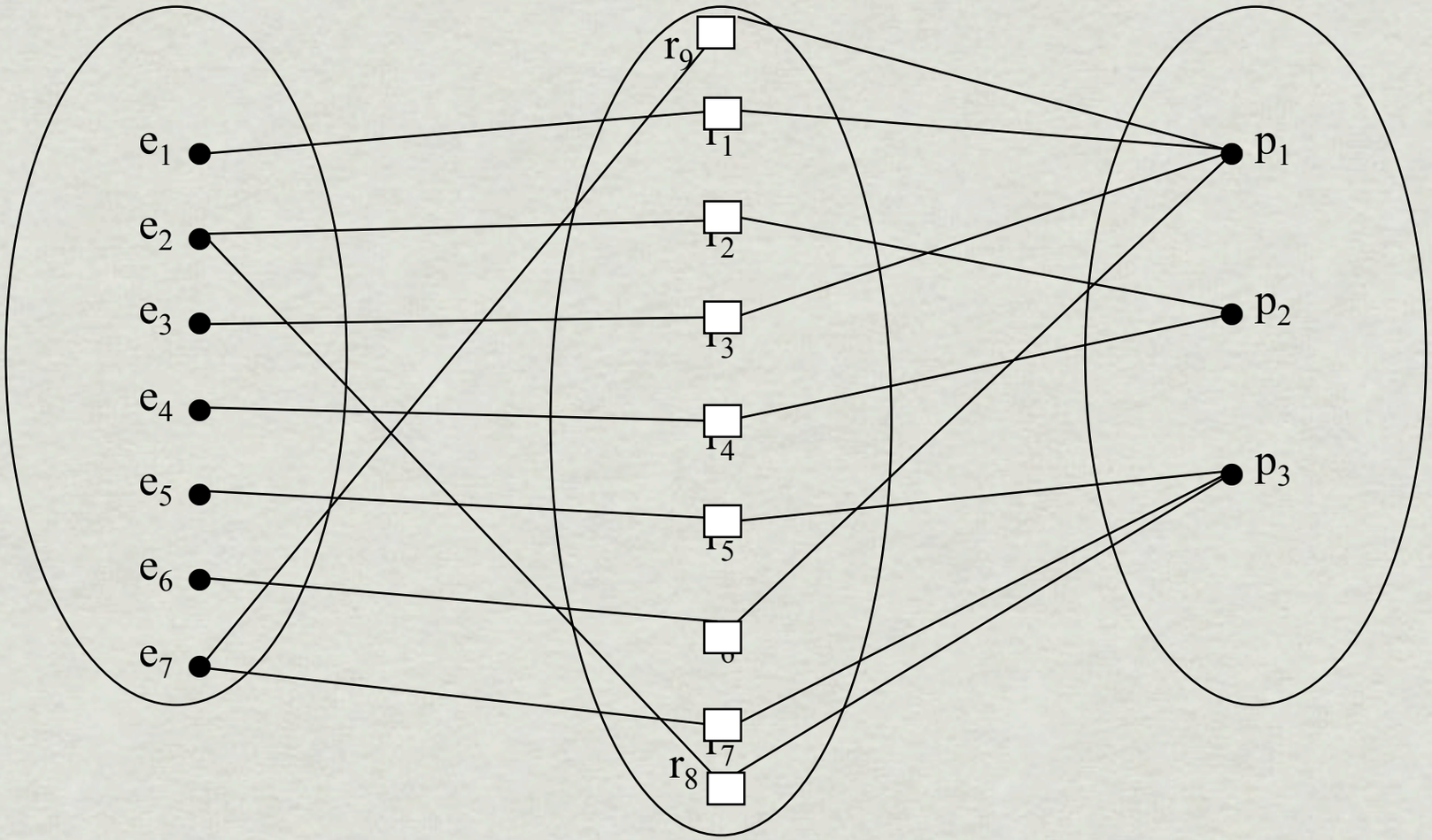
- Relationships of the same type are grouped or typed into a relationship type. For example, the WORKS_ON relationship type in which EMPLOYEEs and PROJECTs participate, or the MANAGES relationship type in which EMPLOYEEs and DEPARTMENTs participate.

- The degree of a relationship type is the number of participating entity types. Both MANAGES and WORKS_ON are binary relationships.

Example relationship instances of the WORKS_FOR relationship between EMPLOYEE and DEPARTMENT



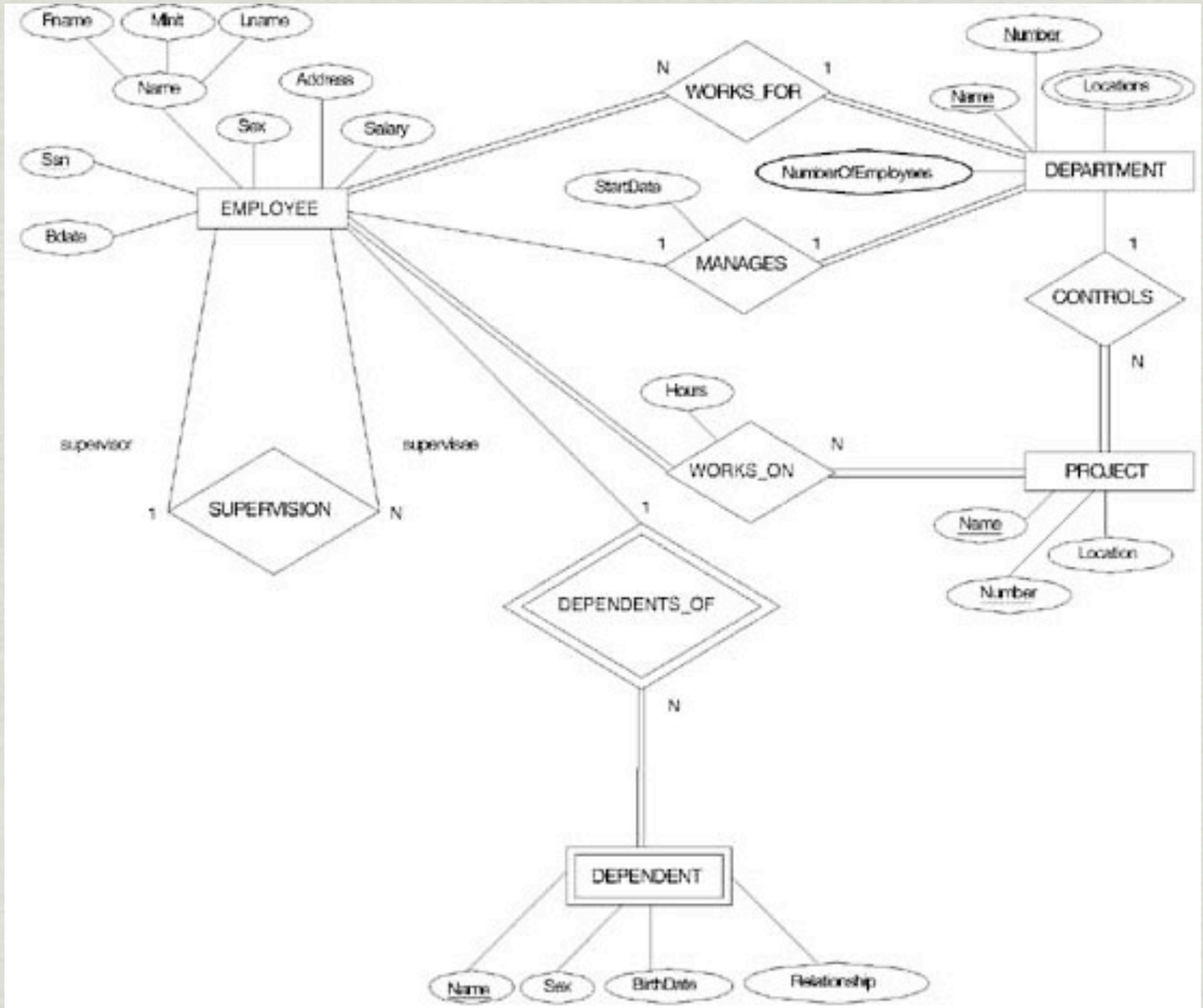
Example relationship instances of the WORKS_ON relationship between EMPLOYEE and PROJECT



Relationships and Relationship Types (2)

- More than one relationship type can exist with the same participating entity types. For example, MANAGES and WORKS_FOR are distinct relationships between EMPLOYEE and DEPARTMENT, but with different meanings and different relationship instances.

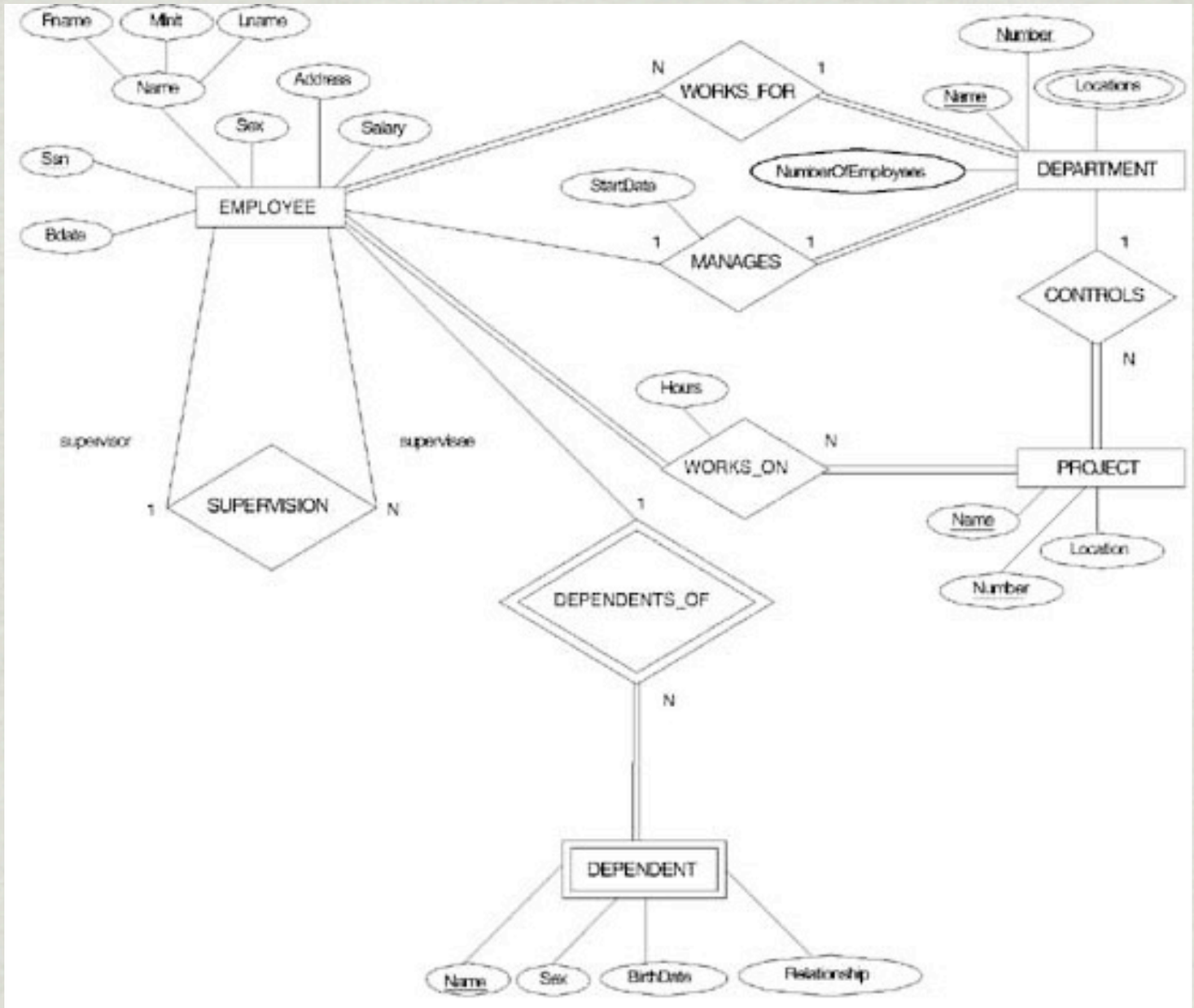
ER DIAGRAM – Relationship Types are: WORKS_FOR, MANAGES, WORKS_ON, CONTROLS, SUPERVISION, DEPENDENTS_OF



Weak Entity Types

- An entity that does not have a key attribute
- A weak entity must participate in an identifying relationship type with an owner or identifying entity type
- Entities are identified by the combination of:
  - A partial key of the weak entity type
  - The particular entity they are related to in the identifying entity type
- Example:
  - Suppose that a DEPENDENT entity is identified by the dependent's first name and birhtdate, and the specific EMPLOYEE that the dependent is related to. DEPENDENT is a weak entity type with EMPLOYEE as its identifying entity type via the identifying relationship type DEPENDENT_OF

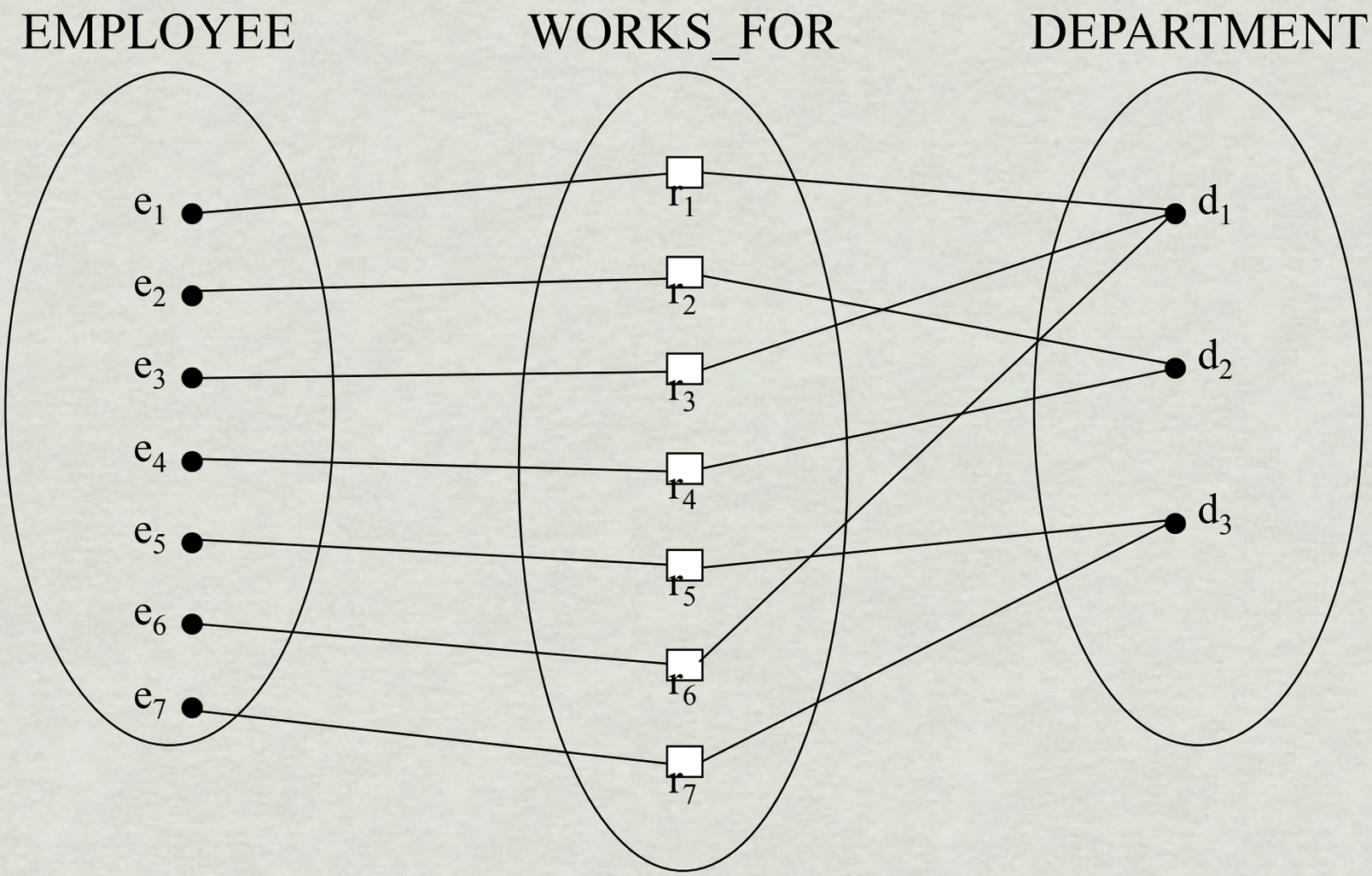
Weak Entity Type is: DEPENDENT Identifying Relationship is: DEPENDENTS_OF



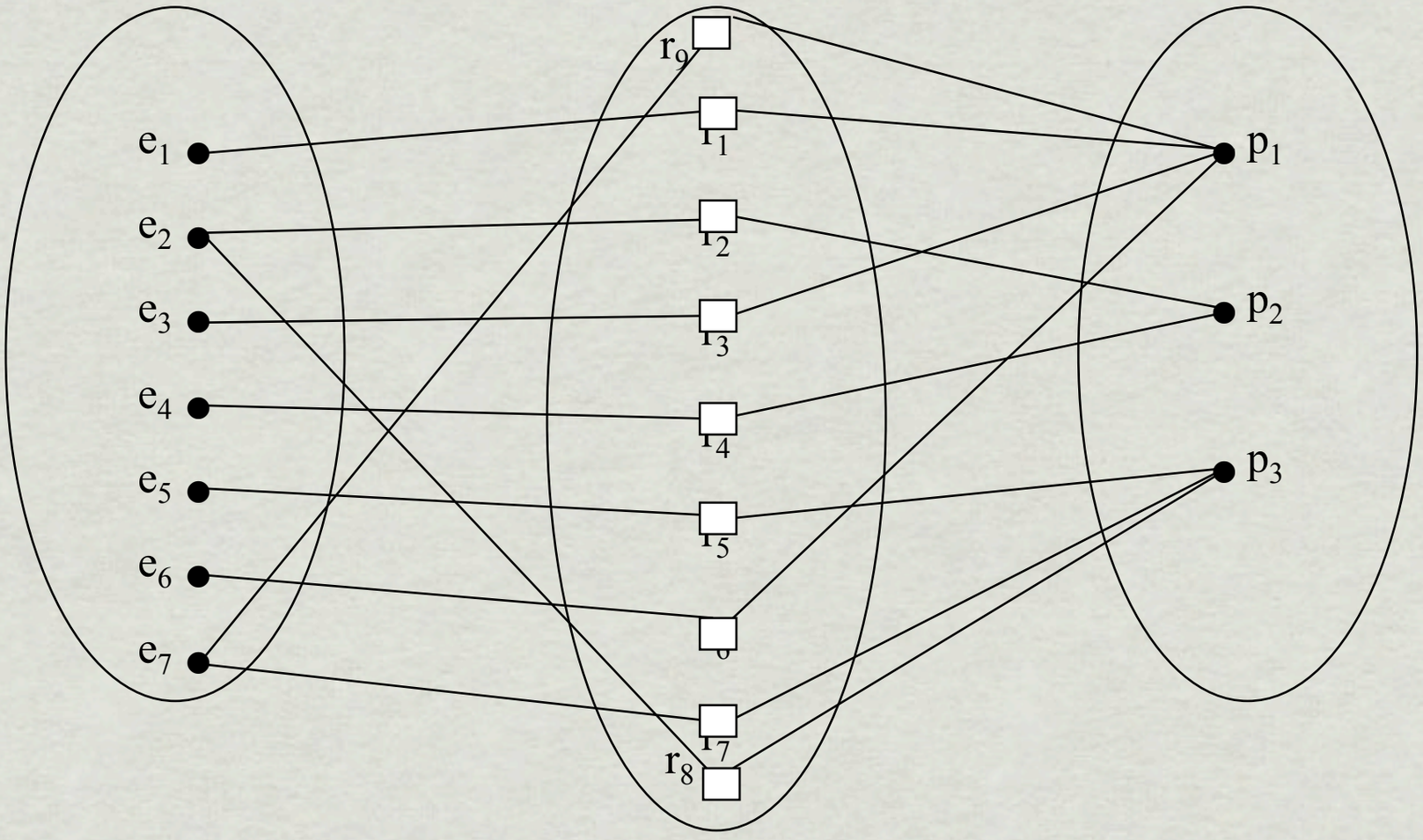
Constraints on Relationships

- Constraints on Relationship Types
  - (Also known as ratio constraints)
  - Maximum Cardinality
    - One-to-one (1:1)
    - One-to-many (1:N) or Many-to-one (N:1)
    - Many-to-many
- Minimum Cardinality (also called participation constraint or existence dependency constraints)
    - zero (optional participation, not existence-dependent)
    - one or more (mandatory, existence-dependent)

Many-to-one (N:1) RELATIONSHIP



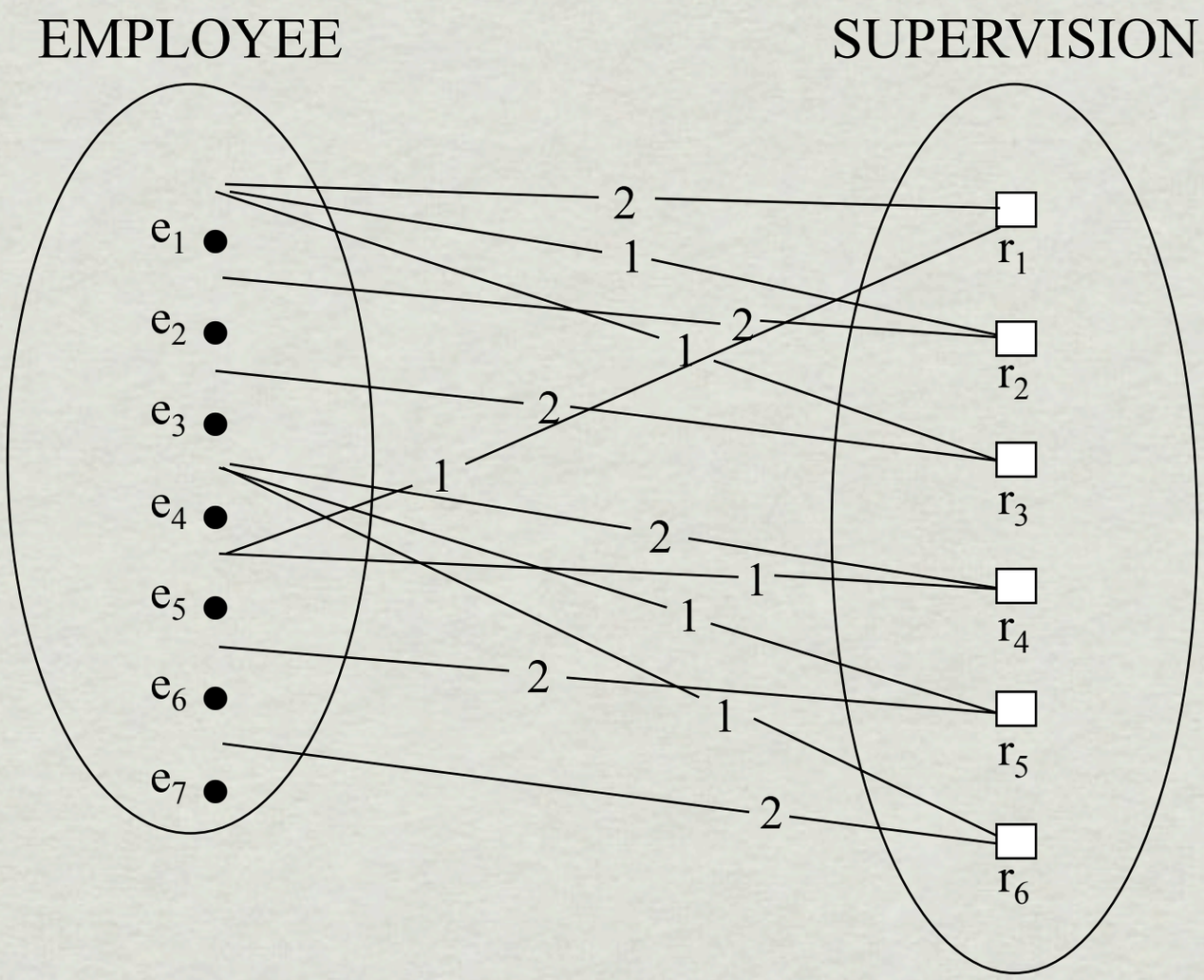
Many-to-many (M:N) RELATIONSHIP



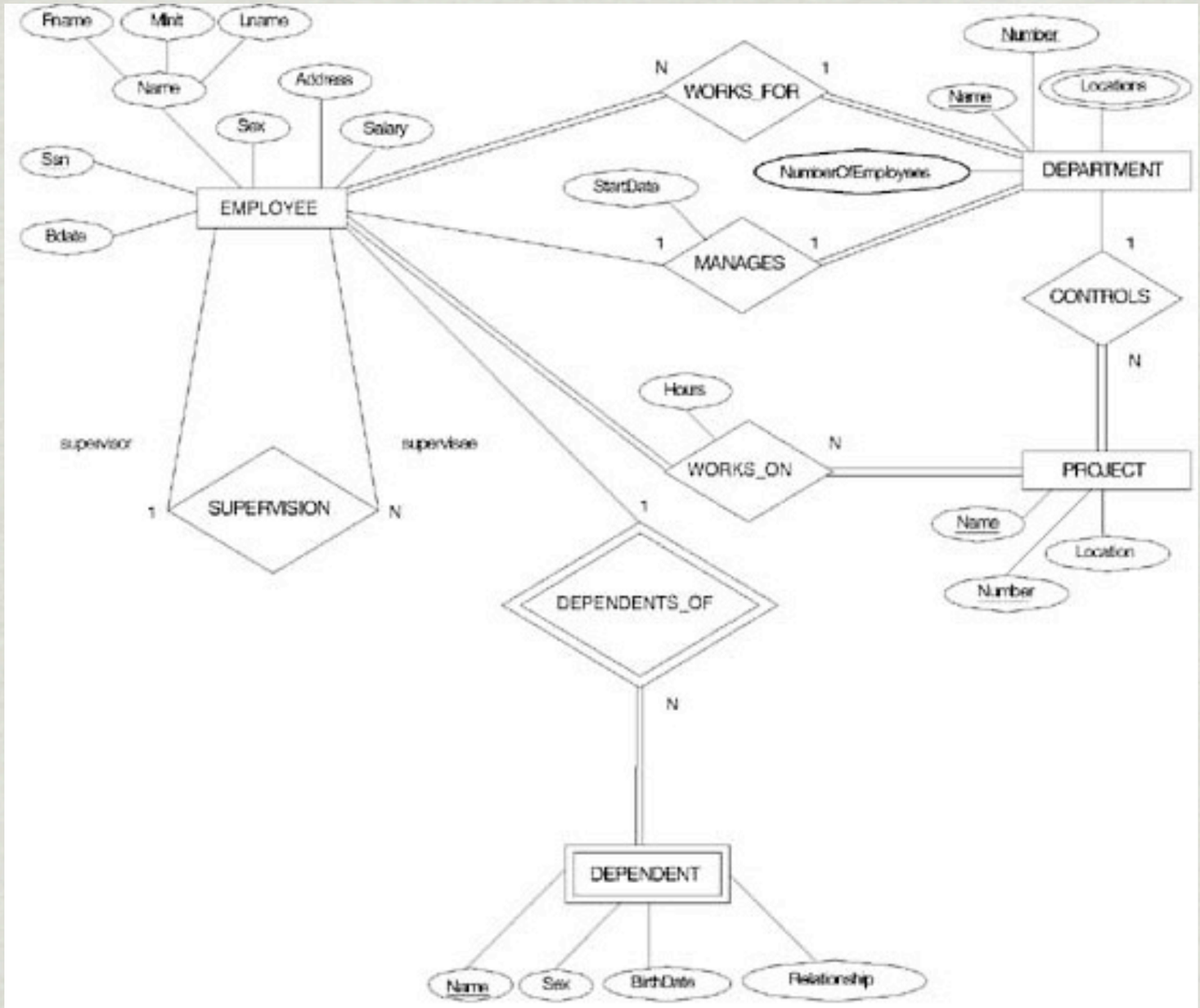
Relationships and Relationship Types (3)

- We can also have a recursive relationship type.
- Both participations are same entity type in different roles.
- For example, SUPERVISION relationships between EMPLOYEE (in role of supervisor or boss) and (another) EMPLOYEE (in role of subordinate or worker).
- In following figure, first role participation labeled with 1 and second role participation labeled with 2.
- In ER diagram, need to display role names to distinguish participations.

A RECURSIVE RELATIONSHIP SUPERVISION



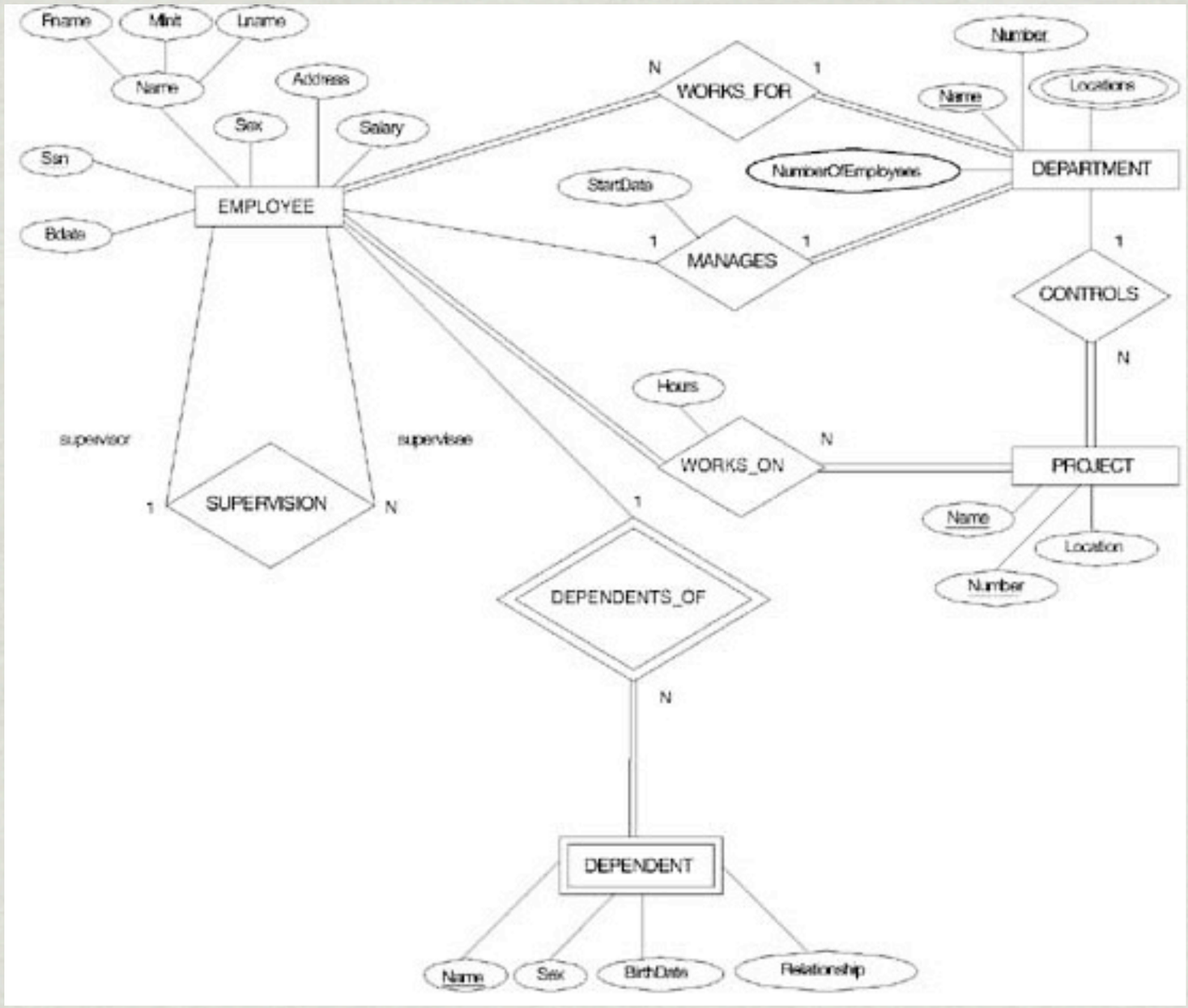
Recursive Relationship Type is: SUPERVISION (participation role names are shown)



Attributes of Relationship types

- A relationship type can have attributes; for example, HoursPerWeek of WORKS_ON; its value for each relationship instance describes the number of hours per week that an EMPLOYEE works on a PROJECT.

Attribute of a Relationship Type is: Hours of WORKS_ON



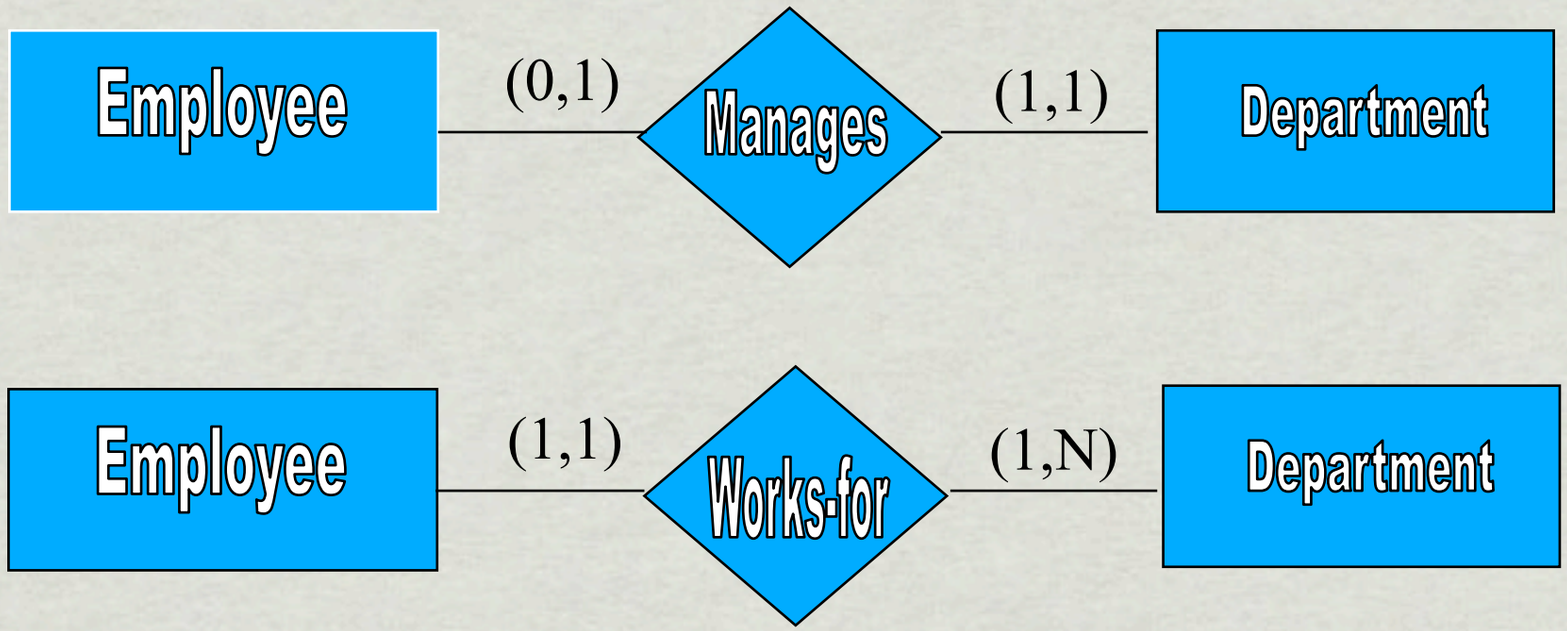
Structural Constraints – one way to express semantics of relationships

- Structural constraints on relationships:
- Cardinality ratio (of a binary relationship): 1:1, 1:N, N:1, or M:N
  - SHOWN BY PLACING APPROPRIATE NUMBER ON THE LINK.
- Participation constraint (on each participating entity type): total (called existence dependency) or partial.
  - SHOWN BY DOUBLE LINING THE LINK
- NOTE: These are easy to specify for Binary Relationship Types.

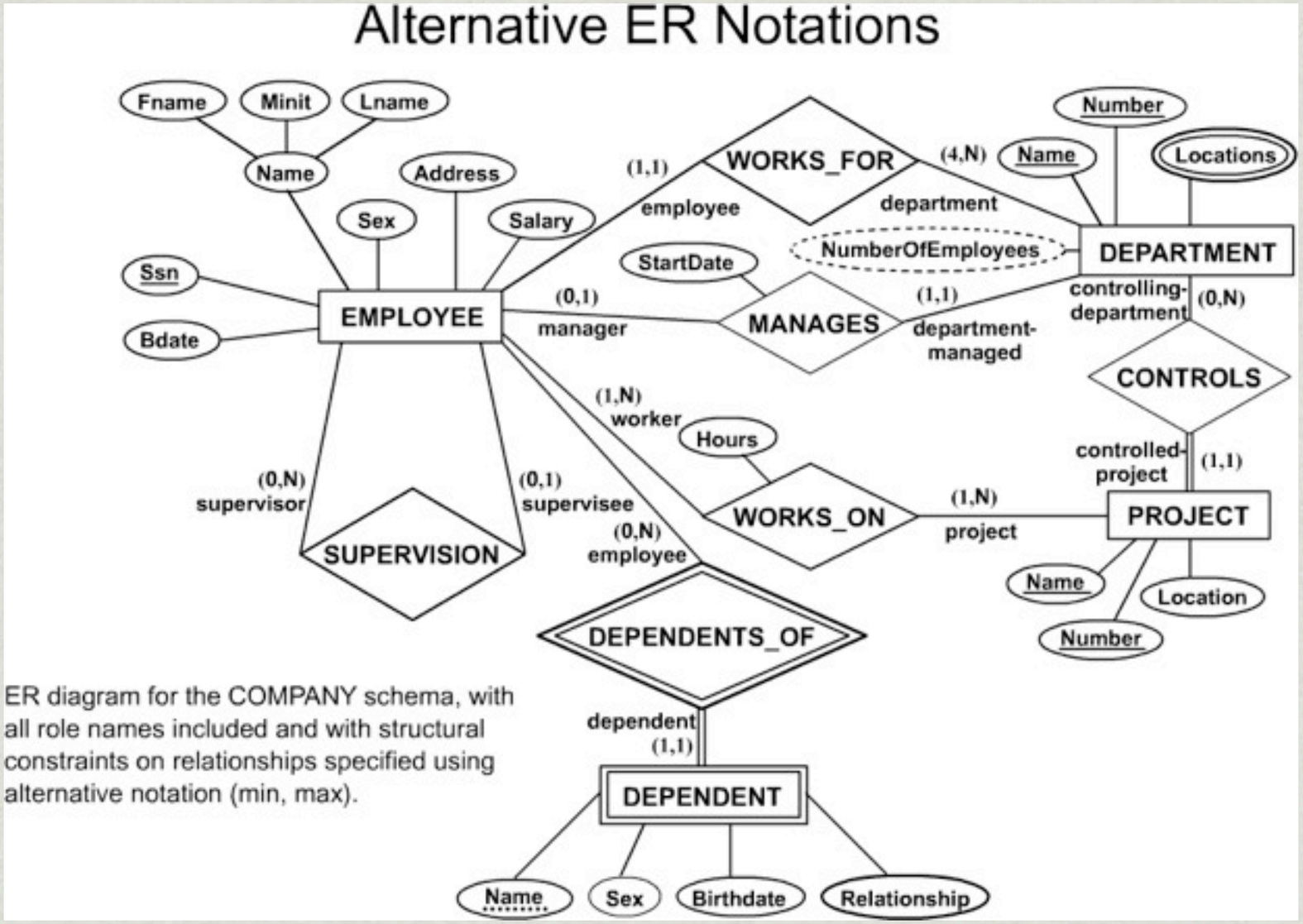
Alternative (min, max) notation for relationship structural constraints:

- Specified on each participation of an entity type E in a relationship type R
- Specifies that each entity e in E participates in at least min and at most max relationship instances in R
- Default(no constraint): min=0, max=n
- Must have min≤max, min≥0, max ≥1
- Derived from the knowledge of mini-world constraints
- Examples:
  - A department has exactly one manager and an employee can manage at most one department.
    - Specify (0,1) for participation of EMPLOYEE in MANAGES
    - Specify (1,1) for participation of DEPARTMENT in MANAGES
  - An employee can work for exactly one department but a department can have any number of employees.
    - Specify (1,1) for participation of EMPLOYEE in WORKS_FOR
    - Specify (0,n) for participation of DEPARTMENT in WORKS_FOR

(min,max) notation



COMPANY ER Schema Diagram using (min, max) notation



ER diagram for the COMPANY schema, with all role names included and with structural constraints on relationships specified using alternative notation (min, max).

Relationships of Higher Degree

- Relationship types of degree 2 are called binary
- Relationship types of degree 3 are called ternary and of degree n are called n-ary
- In general, an n-ary relationship is not equivalent to n binary relationships

PROBLEM with ER notation

- THE ENTITY RELATIONSHIP MODEL IN ITS ORIGINAL FORM DID NOT SUPPORT THE SPECIALIZATION/GENERALIZATION ABSTRACTIONS

Extended Entity-Relationship (EER) Model

- Incorporates Set-subset relationships
- Incorporates Specialization/Generalization Hierarchies

Relational Model

Relational Model

- Relational Model Concepts
- Relational Model Constraints and Relational Database Schemas
- Update Operations and Dealing with Constraint Violations

Concepts

- The relational Model of Data is based on the concept of a Relation.
- A Relation is a mathematical concept based on the ideas of sets.
- The strength of the relational approach to data management comes from the formal foundation provided by the theory of relations.
- We review the essentials of the relational approach in the next slides.

Concepts

- The model was first proposed by Dr. E.F. Codd of IBM in 1970 in the following paper:
  "A Relational Model for Large Shared Data Banks," Communications of the ACM, June 1970.

- The above paper caused a major revolution in the field of Database management and earned Ted Codd the coveted ACM Turing Award.

INFORMAL DEFINITIONS

- RELATION: A table of values
  - A relation may be thought of as a set of rows.
  - A relation may alternately be though of as a set of columns.
  - Each row represents a fact that corresponds to a real-world entity or relationship.
  - Each row has a value of an item or set of items that uniquely identifies that row in the table.
  - Sometimes row-ids or sequential numbers are assigned to identify the rows in the table.
  - Each column typically is called by its column name or column header or attribute name.

FORMAL DEFINITIONS

- A Relation may be defined in multiple ways.
- The Schema of a Relation: R (A1, A2, ..., An)
  - Relation schema R is defined over attributes A1, A2, ..., An
    - For Example:
      CUSTOMER (Cust-id, Cust-name, Address, Phone#)
- Here, CUSTOMER is a relation defined over the four attributes Cust-id, Cust-name, Address, Phone#, each of which has a domain or a set of valid values.
  For example, the domain of Cust-id is 6 digit numbers.

FORMAL DEFINITIONS

- A tuple is an ordered set of values
- Each value is derived from an appropriate domain.
- Each row in the CUSTOMER table may be referred to as a tuple in the table and would consist of four values.
- <632895, "John Smith", "101 Main St. Atlanta, GA 30332", "(404) 894-2000">
  is a tuple belonging to the CUSTOMER relation.
- A relation may be regarded as a set of tuples (rows).
- Columns in a table are also called attributes of the relation.

FORMAL DEFINITIONS

- A domain has a logical definition: e.g., “USA_phone_numbers” are the set of 10 digit phone numbers valid in the U.S.
- A domain may have a data-type or a format defined for it. The USA_phone_numbers may have a format: (ddd)-ddd-dddd where each d is a decimal digit. E.g., Dates have various formats such as monthname, date, year or yyyy-mm-dd, or dd mm,yyyy etc.
- An attribute designates the role played by the domain. E.g., the domain Date may be used to define attributes “Invoice-date” and “Payment-date”.

FORMAL DEFINITIONS

- The relation is formed over the cartesian product of the sets; each set has values from a domain; that domain is used in a specific role which is conveyed by the attribute name.
- For example, attribute Cust-name is defined over the domain of strings of 25 characters. The role these strings play in the CUSTOMER relation is that of the name of customers.
- Formally,
  - Given R(A1, A2, ..., An)
    - r(R) ⊂ dom (A1) X dom (A2) X .... X dom(An)
- R: schema of the relation
- r of R: a specific "value" or population of R.

FORMAL DEFINITIONS

- Let S1 = {0,1}
- Let S2 = {a,b,c}

- Let R ⊂ S1 X S2

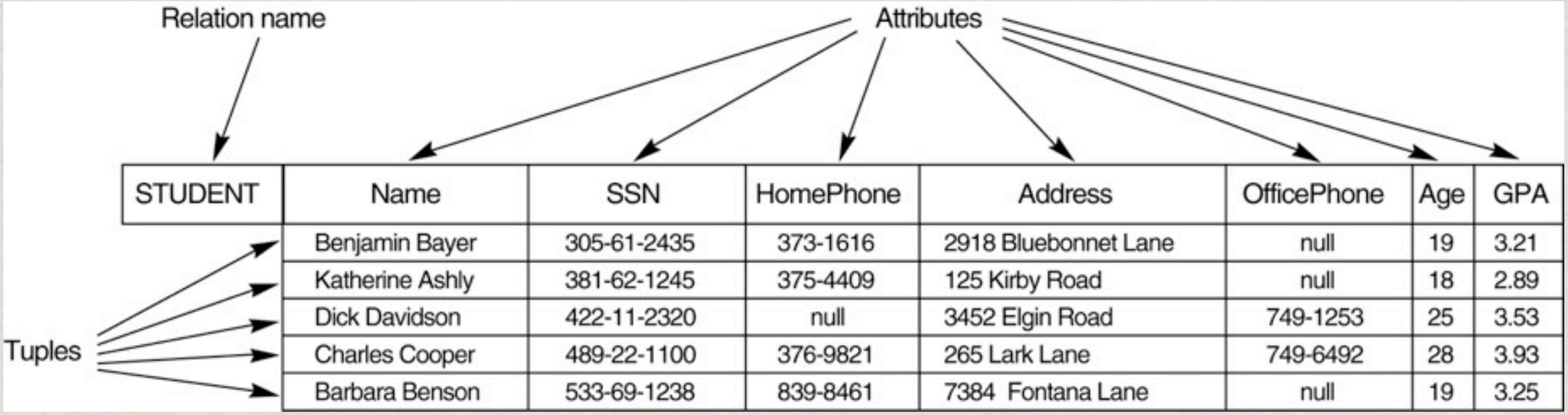
- Then for example: r(R) = {<0,a> , <0,b> , <1,c> }

  is one possible “state” or “population” or “extension” r of the relation R, defined over domains S1 and S2. It has three tuples.

DEFINITION SUMMARY

Informal Terms		Formal Terms
Table		Relation
Column		Attribute/Domain
Row		Tuple
Values in a column		Domain
Table Definition		Schema of a Relation
Populated Table		Extension

Example



STUDENT	Name	SSN	HomePhone	Address	OfficePhone	Age	GPA
	Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	null	19	3.21
	Katherine Ashly	381-62-1245	375-4409	125 Kirby Road	null	18	2.89
	Dick Davidson	422-11-2320	null	3452 Elgin Road	749-1253	25	3.53
	Charles Cooper	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
	Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	null	19	3.25

CHARACTERISTICS OF RELATIONS

- Ordering of tuples in a relation r(R): The tuples are not considered to be ordered, even though they appear to be in the tabular form.
- Ordering of attributes in a relation schema R (and of values within each tuple): We will consider the attributes in R(A1, A2, ..., An) and the values in t=<v1, v2, ..., vn> to be ordered.
- (**However, a more general alternative definition of relation does not require this ordering**). But... Physical level may rely on order!
- Values in a tuple: All values are considered atomic (indivisible). A special null value is used to represent values that are unknown or inapplicable to certain tuples.

CHARACTERISTICS OF RELATIONS

- Notation:
- We refer to component values of a tuple t by t[Ai] = vi (the value of attribute Ai for tuple t).
- Similarly, t[Au, Av, ..., Aw] refers to the subtuple of t containing the values of attributes Au, Av, ..., Aw, respectively.

Relational Integrity Constraints

- Constraints are conditions that must hold on all valid relation instances. There are three main types of constraints:
  - Key constraints
  - Entity integrity constraints
  - Referential integrity constraints

Key Constraints

- **Superkey** of R: A set of attributes SK of R such that no two tuples in any valid relation instance r(R) will have the same value for SK. That is, for any distinct tuples t1 and t2 in r(R), t1[SK] ≠ t2[SK].
- **Key** of R: A "minimal" superkey; that is, a superkey K such that removal of any attribute from K results in a set of attributes that is not a superkey.
  - Example: The CAR relation schema:
    - CAR(State, Reg#, SerialNo, Make, Model, Year)
    - has two keys, Key1 = {State, Reg#}, Key2 = {SerialNo}, which are also superkeys. {SerialNo, Make} is a superkey but not a key.
- If a relation has several candidate keys, one is chosen arbitrarily to be the **primary key**. The primary key attributes are underlined.

Key Constraints

Figure 7.4 The CAR relation with two candidate keys: LicenseNumber and EngineSerialNumber.

CAR	LicenseNumber	EngineSerialNumber	Make	Model	Year
	Texas ABC-739	A69352	Ford	Mustang	96
	Florida TVP-347	B43696	Oldsmobile	Cutlass	99
	New York MPO-22	X83554	Oldsmobile	Delta	95
	California 432-TFY	C43742	Mercedes	190-D	93
	California RSK-629	Y82935	Toyota	Camry	98
	Texas RSK-629	U028365	Jaguar	XJS	98

Entity Integrity

- Relational **Database Schema**: A set S of relation schemas that belong to the same database. S is the name of the database.
- S = {R1, R2, ..., Rn}
- **Entity Integrity**: The primary key attributes PK of each relation schema R in S cannot have null values in any tuple of r(R). This is because primary key values are used to identify the individual tuples.
- t[PK] ≠ null for any tuple t in r(R)
- Note: Other attributes of R may be similarly constrained to disallow null values, even though they are not members of the primary key.

Referential Integrity

- A constraint involving two relations (the previous constraints involve a single relation).
- Used to specify a relationship among tuples in two relations: the **referencing** relation and the **referenced** relation.
- Tuples in the referencing relation R1 have attributes FK (called foreign key attributes) that reference the primary key attributes PK of the referenced relation R2. A tuple t1 in R1 is said to reference a tuple t2 in R2 if t1[FK] = t2[PK].
- A referential integrity constraint can be displayed in a relational database schema as a directed arc from R1.FK to R2.

Referential Integrity Constraint

- Statement of the constraint
- The value in the foreign key column (or columns) FK of the the referencing relation R1 can be either:

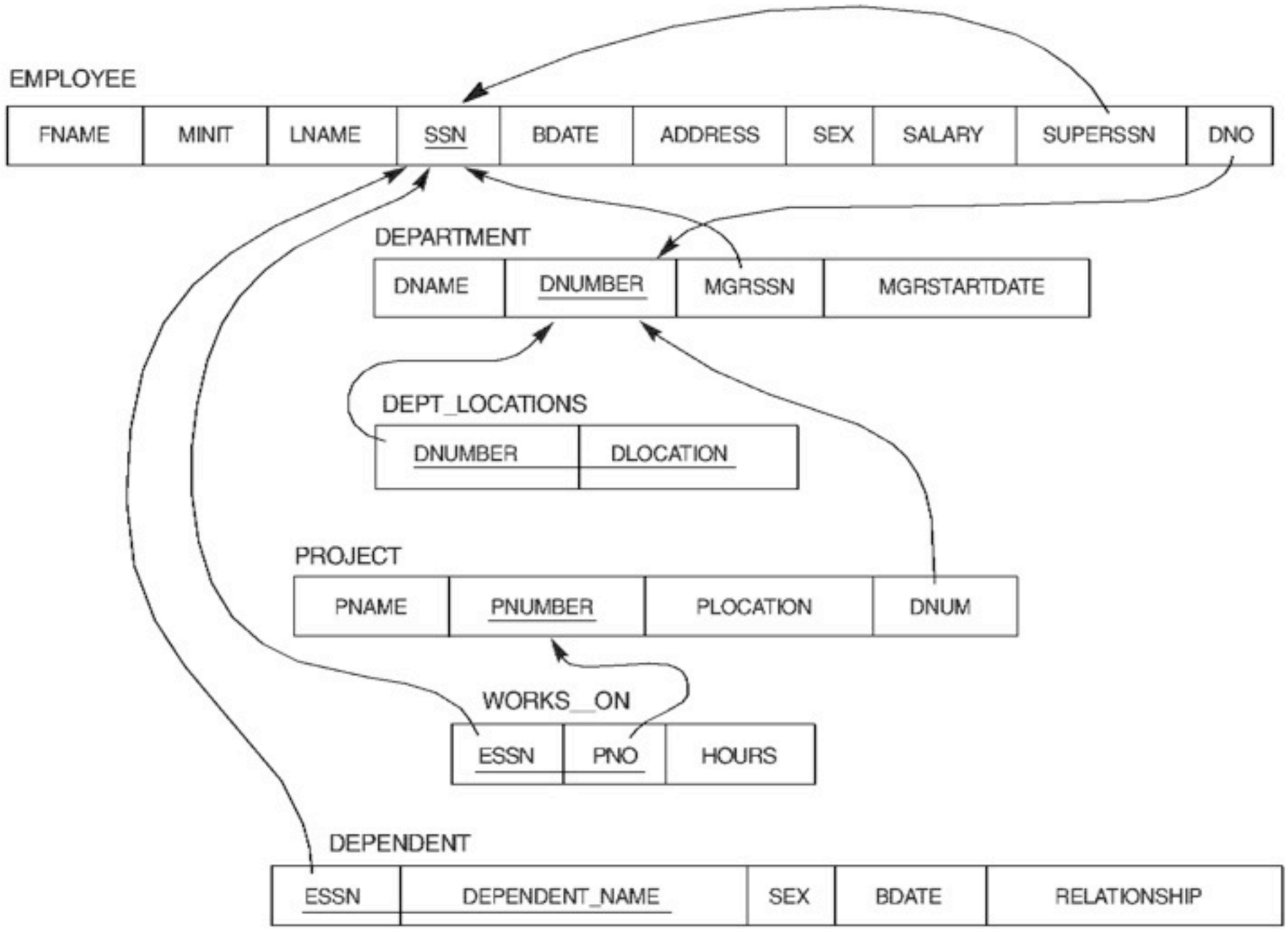
  1. a value of an existing primary key value of the corresponding primary key PK in the referenced relation R2, or
  2. a null.

- In case (2), the FK in R1 should not be a part of its own primary key.

Other Types of Constraints

- Semantic Integrity Constraints:
- based on application semantics and cannot be expressed by the model per se
- E.g., “the max. no. of hours per employee for all projects he or she works on is 56 hrs per week”
- A constraint specification language may have to be used to express these
- SQL-99 allows TRIGGERS and ASSERTIONS to allow for some of these

Figure 7.7 Referential integrity constraints displayed on the COMPANY relational database schema diagram.



Update Operations on Relations

- INSERT a tuple.
- DELETE a tuple.
- MODIFY a tuple.

- Integrity constraints should not be violated by the update operations.
- Several update operations may have to be grouped together.
- Updates may propagate to cause other updates automatically. This may be necessary to maintain integrity constraints.

Update Operations on Relations

- In case of integrity violation, several actions can be taken:
  - Cancel the operation that causes the violation (REJECT option)
  - Perform the operation but inform the user of the violation
  - Trigger additional updates so the violation is corrected (CASCADE option, SET NULL option)
  - Execute a user-specified error-correction routine

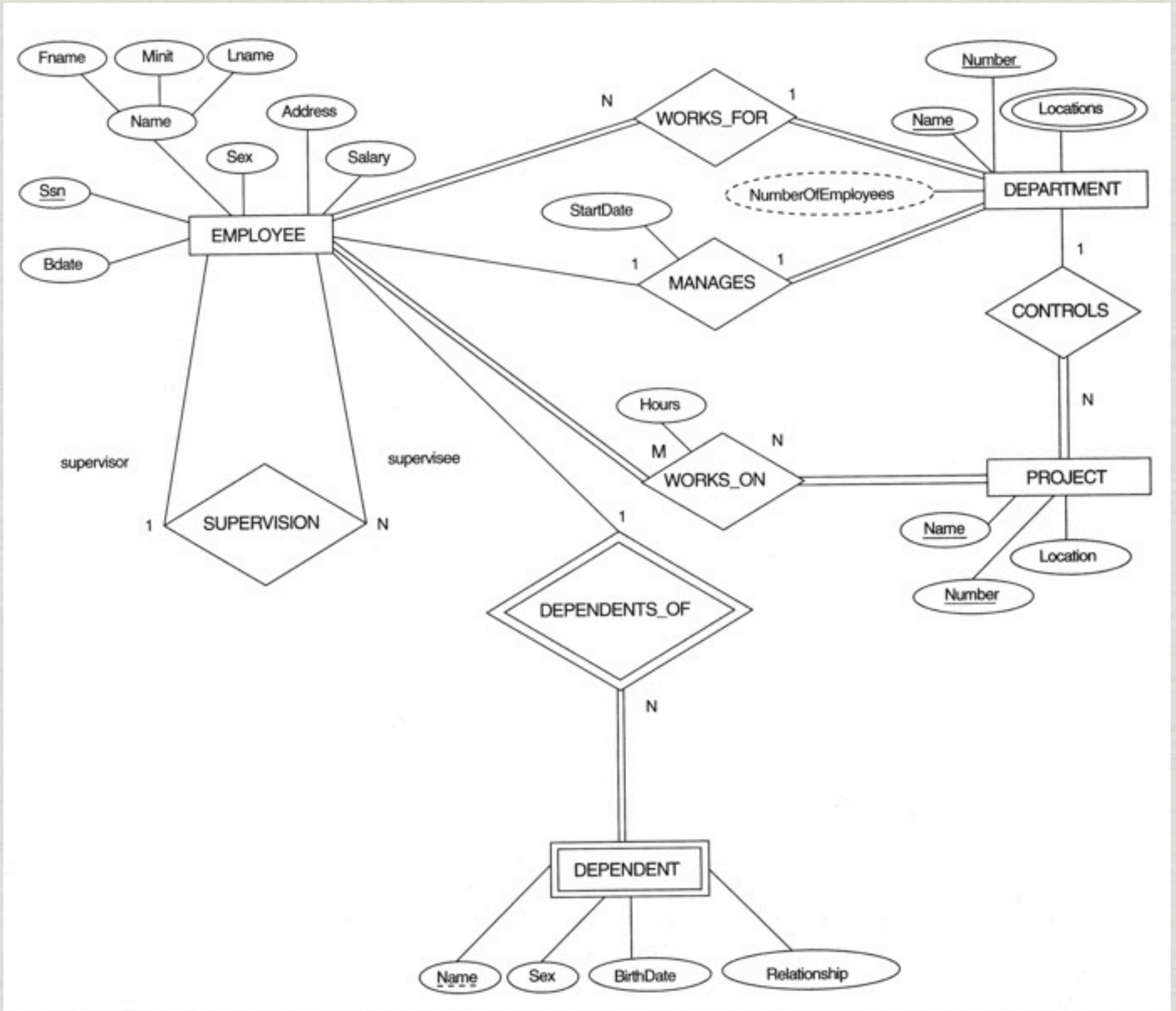
Conceptual – Relational model Mapping

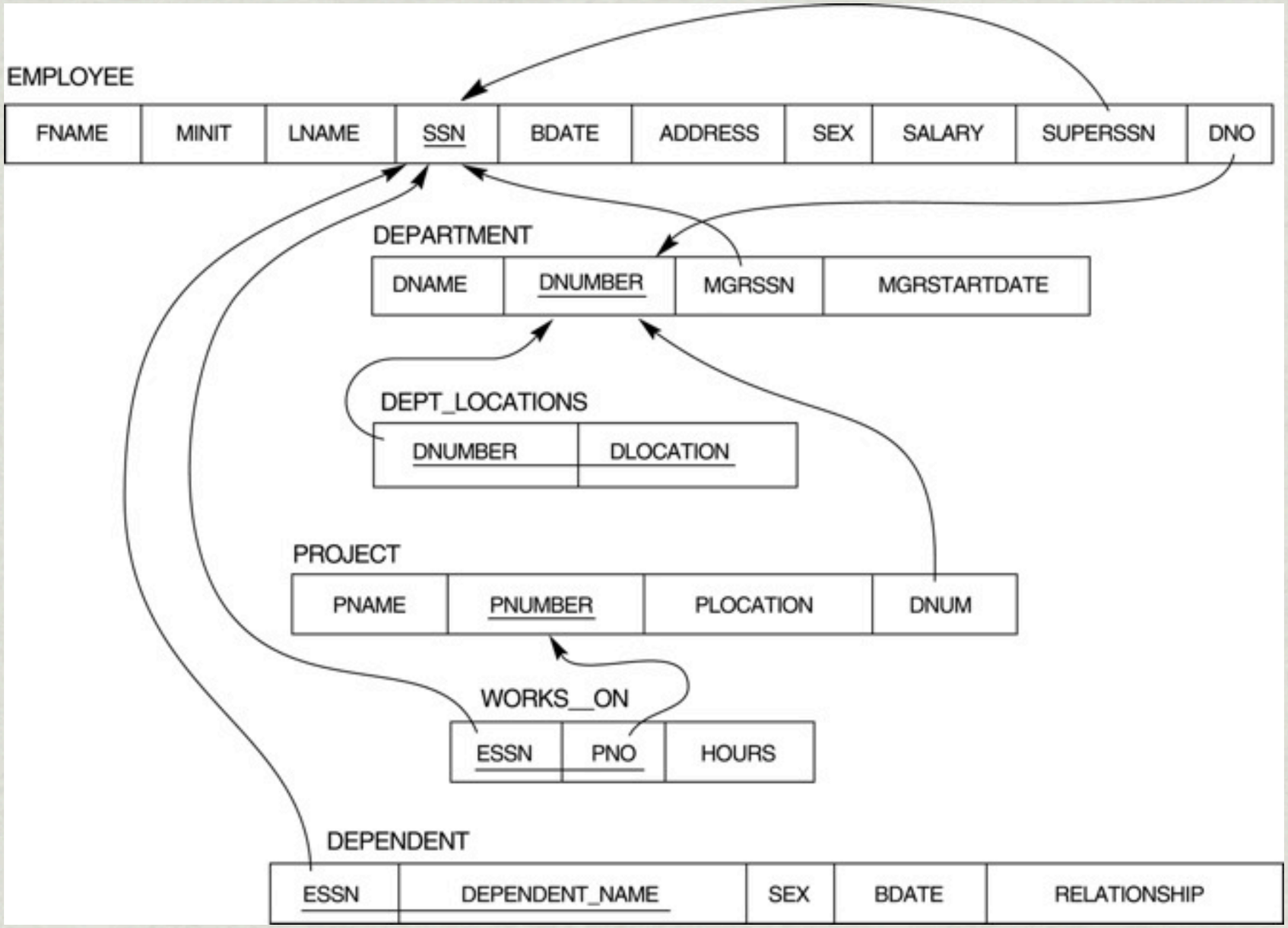
ER – Relational Mapping

- ER-to-Relational Mapping Algorithm
  - Step 1: Mapping of Regular Entity Types
  - Step 2: Mapping of Weak Entity Types
  - Step 3: Mapping of Binary 1:1 Relation Types
  - Step 4: Mapping of Binary 1:N Relationship Types.
  - Step 5: Mapping of Binary M:N Relationship Types.
  - Step 6: Mapping of Multivalued attributes.
  - Step 7: Mapping of N-ary Relationship Types.
- Mapping EER Model Constructs to Relations
  - Step 8: Options for Mapping Specialization or Generalization.
  - Step 9: Mapping of Union Types (Categories).

ER-to-Relational Mapping Algorithm

- Step 1: Mapping of Regular Entity Types.
  - For each regular (strong) entity type E in the ER schema, create a relation R that includes all the simple attributes of E.
  - Choose one of the key attributes of E as the primary key for R. If the chosen key of E is composite, the set of simple attributes that form it will together form the primary key of R.
    - Example: We create the relations EMPLOYEE, DEPARTMENT, and PROJECT in the relational schema corresponding to the regular entities in the ER diagram. SSN, DNUMBER, and PNUMBER are the primary keys for the relations EMPLOYEE, DEPARTMENT, and PROJECT as shown.


Fname
Minit
Lname
Name
Address
Sex
Salary
Ssn
Bdate
EMPLOYEE
N
WORKS_FOR
1
Number
Name
Locations
DEPARTMENT
NumberOfEmployees
StartDate
1
MANAGES
1
1
CONTROLS
N
Hours
M
N
WORKS_ON
PROJECT
Name
Location
Number
supervisor
supervisee
1
SUPERVISION
N
1
DEPENDENTS_OF
N
DEPENDENT
Name
Sex
BirthDate
Relationship


EMPLOYEE
FNAME
MINIT
LNAME
SSN
BDATE
ADDRESS
SEX
SALARY
SUPERSSN
DNO
DEPARTMENT
DNAME
DNUMBER
MGRSSN
MGRSTARTDATE
DEPT_LOCATIONS
DNUMBER
DLOCATION
PROJECT
PNAME
PNUMBER
PLOCATION
DNUM
WORKS_ON
ESSN
PNO
HOURS
DEPENDENT
ESSN
DEPENDENT_NAME
SEX
BDATE
RELATIONSHIP

ER-to-Relational Mapping Algorithm

- Step 2: Mapping of Weak Entity Types
  - For each weak entity type W in the ER schema with owner entity type E, create a relation R and include all simple attributes (or simple components of composite attributes) of W as attributes of R.
  - In addition, include as foreign key attributes of R the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s).
  - The primary key of R is the combination of the primary key(s) of the owner(s) and the partial key of the weak entity type W, if any.
    - Example: Create the relation DEPENDENT in this step to correspond to the weak entity type DEPENDENT. Include the primary key SSN of the EMPLOYEE relation as a foreign key attribute of DEPENDENT (renamed to ESSN).
  - The primary key of the DEPENDENT relation is the combination {ESSN, DEPENDENT_NAME} because DEPENDENT_NAME is the partial key of DEPENDENT.

ER-to-Relational Mapping Algorithm

- Step 3: Mapping of Binary 1:1 Relation Types
  - For each binary 1:1 relationship type R in the ER schema, identify the relations S and T that correspond to the entity types participating in R. There are three possible approaches:

1. Foreign Key approach: Choose one of the relations-S, say-and include a foreign key in S the primary key of T. It is better to choose an entity type with total participation in R in the role of S.
   - Example: 1:1 relation MANAGES is mapped by choosing the participating entity type DEPARTMENT to serve in the role of S, because its participation in the MANAGES relationship type is total.
2. Merged relation option: An alternate mapping of a 1:1 relationship type is possible by merging the two entity types and the relationship into a single relation. This may be appropriate when both participations are total.
3. Cross-reference or relationship relation option: The third alternative is to set up a third relation R for the purpose of cross-referencing the primary keys of the two relations S and T representing the entity types.

ER-to-Relational Mapping Algorithm

- Step 4: Mapping of Binary 1:N Relationship Types.
  - For each regular binary 1:N relationship type R, identify the relation S that represent the participating entity type at the N-side of the relationship type.
  - Include as foreign key in S the primary key of the relation T that represents the other entity type participating in R.
  - Include any simple attributes of the 1:N relation type as attributes of S.
    - Example: 1:N relationship types WORKS_FOR, CONTROLS, and SUPERVISION in the figure. For WORKS_FOR we include the primary key DNUMBER of the DEPARTMENT relation as foreign key in the EMPLOYEE relation and call it DNO.

ER-to-Relational Mapping Algorithm

- Step 5: Mapping of Binary M:N Relationship Types.
  - For each regular binary M:N relationship type R, create a new relation S to represent R.
  - Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types; their combination will form the primary key of S.
  - Also include any simple attributes of the M:N relationship type (or simple components of composite attributes) as attributes of S.
    - Example: The M:N relationship type WORKS_ON from the ER diagram is mapped by creating a relation WORKS_ON in the relational database schema. The primary keys of the PROJECT and EMPLOYEE relations are included as foreign keys in WORKS_ON and renamed PNO and ESSN, respectively.
    - Attribute HOURS in WORKS_ON represents the HOURS attribute of the relation type. The primary key of the WORKS_ON relation is the combination of the foreign key attributes {ESSN, PNO}.

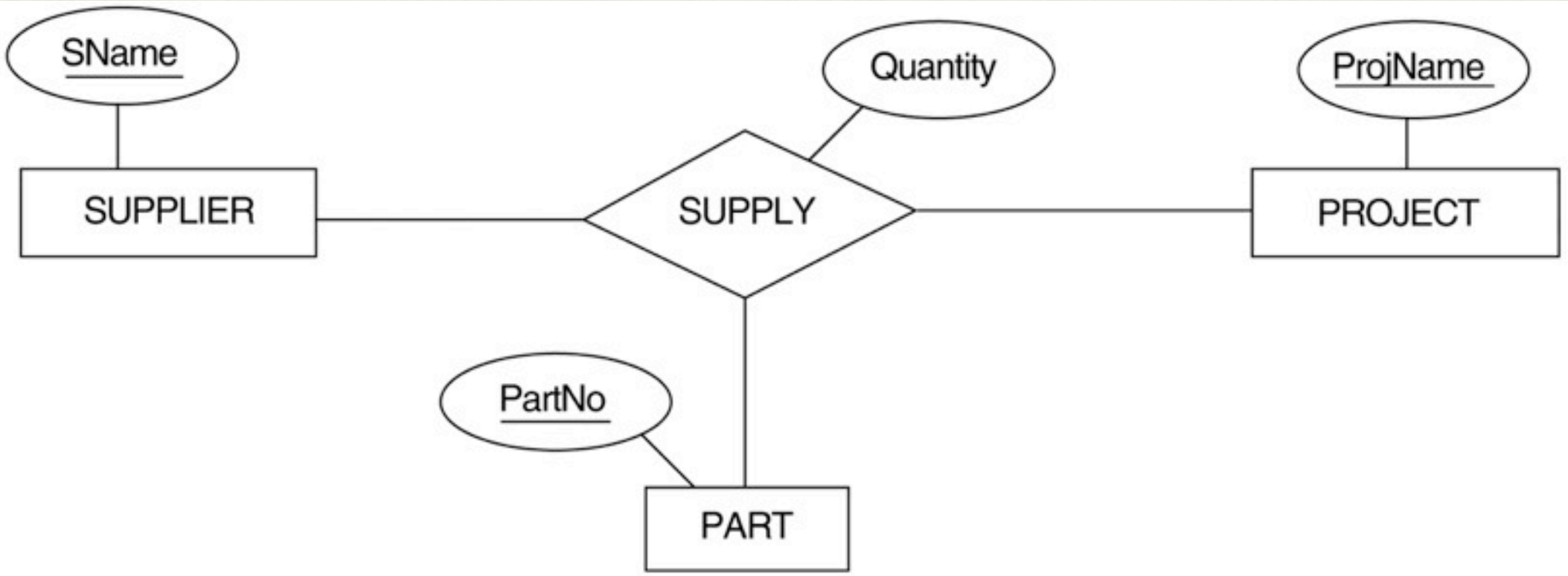
ER-to-Relational Mapping Algorithm

- Step 6: Mapping of Multivalued attributes.
  - For each multivalued attribute A, create a new relation R. This relation R will include an attribute corresponding to A, plus the primary key attribute K-as a foreign key in R-of the relation that represents the entity type of relationship type that has A as an attribute.
  - The primary key of R is the combination of A and K. If the multivalued attribute is composite, we include its simple components.
    - Example: The relation DEPT_LOCATIONS is created. The attribute DLOCATION represents the multivalued attribute LOCATIONS of DEPARTMENT, while DNUMBER-as foreign key-represents the primary key of the DEPARTMENT relation. The primary key of R is the combination of {DNUMBER, DLOCATION}.

ER-to-Relational Mapping Algorithm

- Step 7: Mapping of N-ary Relationship Types.
  - For each n-ary relationship type R, where n>2, create a new relationship S to represent R.
  - Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types.
  - Also include any simple attributes of the n-ary relationship type (or simple components of composite attributes) as attributes of S.
    - Example: The relationship type SUPPY in the ER below. This can be mapped to the relation SUPPLY shown in the relational schema, whose primary key is the combination of the three foreign keys {SNAME, PARTNO, PROJNAME}

(a)


SName
Quantity
ProjName
SUPPLIER
SUPPLY
PROJECT
PartNo
PART

SUPPLIER

SNAME	. . .

PROJECT

PROJNAME	. . .

PART

PARTNO	. . .

SUPPLY

SNAME	PROJNAME	PARTNO	QUANTITY

Summary of Mapping constructs and constraints

ER Model	Relational Model
Entity type	“Entity” Relation
1:1 or 1:N Relationship type	Foreign Key (or “Relationship” relation)
M:N Relationship type	“Relationship” relation and two Foreign Keys
N-ary Relationship type	“Relationship” relation and N Foreign Keys
Simple Attribute	Attribute
Composite Attribute	Set of simple component attributes
Multivalued attribute	Relation and Foreign Key
Value set	Domain
Key Attribute	Primary Key (or Key)

Relational Algebra

Relational Algebra

- Example Database Application (COMPANY)
- Relational Algebra
  - Unary Relational Operations
  - Relational Algebra Operations From Set Theory
  - Binary Relational Operations
  - Additional Relational Operations
  - Examples of Queries in Relational Algebra

Database State for COMPANY

Figure 7.5 Schema diagram for the COMPANY relational database schema; the primary keys are underlined.

EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

DEPARTMENT

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE

DEPT_LOCATIONS

DNUMBER	DLOCATION

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM

WORKS_ON

ESSN	PNO	HOURS

DEPENDENT

ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP

Relational Algebra

- The basic set of operations for the relational model is known as the relational algebra. These operations enable a user to specify basic retrieval requests.
- The result of a retrieval is a new relation, which may have been formed from one or more relations. The algebra operations thus produce new relations, which can be further manipulated using operations of the same algebra.
- A sequence of relational algebra operations forms a relational algebra expression, whose result will also be a relation that represents the result of a database query (or retrieval request).

Unary Relational Operations

- SELECT Operation
- SELECT operation is used to select a subset of the tuples from a relation that satisfy a selection condition. It is a filter that keeps only those tuples that satisfy a qualifying condition – those satisfying the condition are selected while others are discarded.
  - Example: To select the EMPLOYEE tuples whose department number is four or those whose salary is greater than $30,000 the following notation is used:
    - $\sigma_{DNO = 4}$ (EMPLOYEE)
    - $\sigma_{SALARY > 30,000}$ (EMPLOYEE)
- In general, the select operation is denoted by σ <selection condition>(R) where the
- symbol σ (sigma) is used to denote the select operator, and the selection condition is a Boolean expression specified on the attributes of relation R

Unary Relational Operations

- SELECT Operation Properties
- The SELECT operation σ <selection condition>(R) produces a relation S that has the same schema as R
  - The SELECT operation σ is commutative; i.e.,
    - $\sigma_{<condition1>}(\sigma_{<condition2>}(R)) = \sigma_{<condition2>}(\sigma_{<condition1>}(R))$
  - A cascaded SELECT operation may be applied in any order; i.e.,
    - $\sigma_{<condition1>}(\sigma_{<condition2>}(\sigma_{<condition3>}(R)) = \sigma_{<condition2>}(\sigma_{<condition3>}(\sigma_{<condition1>}(R)))$
  - A cascaded SELECT operation may be replaced by a single selection with a conjunction of all the conditions; i.e.,
    - $\sigma_{<condition1>}(\sigma_{<condition2>}(\sigma_{<condition3>}(R)) = \sigma_{<condition1> \text{ AND } <condition2> \text{ AND } <condition3>}(R)$

Figure 7.8 Results of SELECT and PROJECT operations.

(a) $\sigma_{(DNO=4\ AND\ SALARY>25000)\ OR\ (DNO=5\ AND\ SALARY>30000)}(EMPLOYEE)$.

(b) $\pi_{LNAME,\ FNAME,\ SALARY}(EMPLOYEE)$. (c) $\pi_{SEX,\ SALARY}(EMPLOYEE)$

(a)

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
Franklin	T	Wong	333445555	1955-12-08	638 Voss,Houston,TX	M	40000	888665555	5
Jennifer		Wallace	987654321	1941-06-20	291 Berry,Bellaire,TX	F	43000	888665555	4
Ramesh		Narayan	666884444	1962-09-15	975 FireOak,Humble,TX	M	38000	333445555	5

(b)

LNAME	FNAME	SALARY
Smith	John	30000
Wong	Franklin	40000
Zelaya	Alicia	25000
Wallace	Jennifer	43000
Narayan	Ramesh	38000
English	Joyce	25000
Jabbar	Ahmad	25000
Borg	James	55000

(c)

SEX	SALARY
M	30000
M	40000
F	25000
F	43000
M	38000
M	25000
M	55000

Unary Relational Operations (cont.)

- PROJECT Operation
- This operation selects certain columns from the table and discards the other columns. The PROJECT creates a vertical partitioning – one with the needed columns (attributes) containing results of the operation and other containing the discarded Columns.
  - Example: To list each employee's first and last name and salary, the following is used:
    - $\pi_{LNAME, FNAME, SALARY}(EMPLOYEE)$
- The general form of the project operation is $\pi_{<attribute\ list>}(R)$ where π (pi) is the symbol used to represent the project operation and <attribute list> is the desired list of attributes from the attributes of relation R.
- The project operation removes any duplicate tuples, so the result of the project operation is a set of tuples and hence a valid relation.

Unary Relational Operations (cont.)

- PROJECT Operation Properties
  - The number of tuples in the result of projection π <list> (R)is always less or equal to the number of tuples in R.
  - If the list of attributes includes a key of R, then the number of tuples is equal to the number of tuples in R.
    - π <list1> (π <list2> (R)) = π <list1> (R)
      as long as <list2> contains the attributes in <list2>

Figure 7.8 Results of SELECT and PROJECT operations.

(a) $\sigma_{(DNO=4\ AND\ SALARY>25000)\ OR\ (DNO=5\ AND\ SALARY>30000)}(EMPLOYEE)$.

(b) $\pi_{LNAME,\ FNAME,\ SALARY}(EMPLOYEE)$. (c) $\pi_{SEX,\ SALARY}(EMPLOYEE)$

(a)

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
Franklin	T	Wong	333445555	1955-12-08	638 Voss,Houston,TX	M	40000	888665555	5
Jennifer		Wallace	987654321	1941-06-20	291 Berry,Bellaire,TX	F	43000	888665555	4
Ramesh		Narayan	666884444	1962-09-15	975 FireOak,Humble,TX	M	38000	333445555	5

(b)

LNAME	FNAME	SALARY
Smith	John	30000
Wong	Franklin	40000
Zelaya	Alicia	25000
Wallace	Jennifer	43000
Narayan	Ramesh	38000
English	Joyce	25000
Jabbar	Ahmad	25000
Borg	James	55000

(c)

SEX	SALARY
M	30000
M	40000
F	25000
F	43000
M	38000
M	25000
M	55000

Unary Relational Operations (cont.)

- Rename Operation
- We may want to apply several relational algebra operations one after the other. Either we can write the operations as a single relational algebra expression by nesting the operations, or we can apply one operation at a time and create intermediate result relations. In the latter case, we must give names to the relations that hold the intermediate results.
  - Example: To retrieve the first name, last name, and salary of all employees who work in department number 5, we must apply a select and a project operation. We can write a single relational algebra expression as follows:
    - $\pi_{FNAME, LNAME, SALARY}(\sigma_{DNO=5}(EMPLOYEE))$
  - OR We can explicitly show the sequence of operations, giving a name to each intermediate relation:
    - DEP5_EMPS ← $\sigma_{DNO=5}$ (EMPLOYEE)
    - RESULT ← $\pi_{FNAME, LNAME, SALARY}$ (DEP5_EMPS)

Unary Relational Operations (cont.)

- Rename Operation (cont.)
- The rename operator is ρ
- The general Rename operation can be expressed by any of the following forms:
  - ρ S (B1, B2, …, Bn) (R) is a renamed relation S based on R with column names B1, B1, …, Bn.
  - ρ S (R) is a renamed relation S based on R (which does not specify column names).
  - ρ (B1, B2, …, Bn) (R) is a renamed relation with column names B1, B1, …, Bn which does not specify a new relation name.

Relational Algebra Operations From Set Theory

- UNION Operation
  - The result of this operation, denoted by R ∪ S, is a relation that includes all tuples that are either in R or in S or in both R and S. Duplicate tuples are eliminated.
    - Example: To retrieve the social security numbers of all employees who either work in department 5 or directly supervise an employee who works in department 5, we can use the union operation as follows:
      - DEP5_EMPS ← $\sigma_{DNO=5}$ (EMPLOYEE)
      - RESULT1 ← π_{SSN} (DEP5_EMPS)
      - RESULT2(SSN) ← $\pi_{SUPERSSN}$(DEP5_EMPS)
      - RESULT ← RESULT1 ∪ RESULT2
  - The union operation produces the tuples that are in either RESULT1 or RESULT2 or both. The two operands must be “type compatible”.

Relational Algebra Operations From Set Theory

- Type Compatibility
  - The operand relations R1(A1, A2, ..., An) and R2(B1, B2, ..., Bn) must have the same number of attributes, and the domains of corresponding attributes must be compatible; that is, dom(Ai)=dom(Bi) for i=1, 2, ..., n.
  - The resulting relation for R1∪R2,R1 ∩ R2, or R1-R2 has the same attribute names as the first operand relation R1 (by convention).

Figure 7.11 Illustrating the set operations union, intersection, and difference. (a) Two union compatible relations. (b) STUDENT ∪ INSTRUCTOR. (c) STUDENT ∪ INSTRUCTOR. (d) STUDENT − INSTRUCTOR. (e) INSTRUCTOR − STUDENT.

(a)

STUDENT	FN	LN
	Susan	Yao
	Ramesh	Shah
	Johnny	Kohler
	Barbara	Jones
	Amy	Ford
	Jimmy	Wang
	Ernest	Gilbert

INSTRUCTOR	FNAME	LNAME
	John	Smith
	Ricardo	Browne
	Susan	Yao
	Francis	Johnson
	Ramesh	Shah

(b)

FN	LN
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

(c)

FN	LN
Susan	Yao
Ramesh	Shah

(d)

FN	LN
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

(e)

FNAME	LNAME
John	Smith
Ricardo	Browne
Francis	Johnson

Relational Algebra Operations From Set Theory (cont.)

- CARTESIAN (or cross product) Operation
  - This operation is used to combine tuples from two relations in a combinatorial fashion. In general, the result of R(A1, A2, . . ., An) x S(B1, B2, . . ., Bm) is a relation Q with degree n + m attributes Q(A1, A2, . . ., An, B1, B2, . . ., Bm), in that order. The resulting relation Q has one tuple for each combination of tuples—one from R and one from S.
  - Hence, if R has nR tuples (denoted as |R| = nR), and S has nS tuples, then
    - | R x S | will have nR * nS tuples.
  - The two operands do NOT have to be "type compatible"
  - Example:
    - FEMALE_EMPS ← $\sigma_{SEX='F'}$(EMPLOYEE)
    - EMPNAMES ← $\pi_{FNAME, LNAME, SSN}$ (FEMALE_EMPS)
    - EMP_DEPENDENTS ← EMPNAMES x DEPENDENT

Figure 7.12 An illustration of the CARTESIAN PRODUCT operation.

FEMALE_EMPS

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
Alicia	J	Zelaya	999887777	1968-07-19	3321 Castle,Spring,TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry,Bellaire,TX	F	43000	888665555	4
Joyce	A	English	453453453	1972-07-31	5631 Rice,Houston,TX	F	25000	333445555	5

EMPNAMES

FNAME	LNAME	SSN
Alicia	Zelaya	999887777
Jennifer	Wallace	987654321
Joyce	English	453453453

EMP_DEPENDENTS

FNAME	LNAME	SSN	ESSN	DEPENDENT_NAME	SEX	BDATE	...
Alicia	Zelaya	999887777	333445555	Alice	F	1986-04-05	...
Alicia	Zelaya	999887777	333445555	Theodore	M	1983-10-25	...
Alicia	Zelaya	999887777	333445555	Joy	F	1958-05-03	...
Alicia	Zelaya	999887777	987654321	Abner	M	1942-02-28	...
Alicia	Zelaya	999887777	123456789	Michael	M	1988-01-04	...
Alicia	Zelaya	999887777	123456789	Alice	F	1988-12-30	...
Alicia	Zelaya	999887777	123456789	Elizabeth	F	1967-05-05	...
Jennifer	Wallace	987654321	333445555	Alice	F	1986-04-05	...
Jennifer	Wallace	987654321	333445555	Theodore	M	1983-10-25	...
Jennifer	Wallace	987654321	333445555	Joy	F	1958-05-03	...
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	...
Jennifer	Wallace	987654321	123456789	Michael	M	1988-01-04	...
Jennifer	Wallace	987654321	123456789	Alice	F	1988-12-30	...
Jennifer	Wallace	987654321	123456789	Elizabeth	F	1967-05-05	...
Joyce	English	453453453	333445555	Alice	F	1986-04-05	...
Joyce	English	453453453	333445555	Theodore	M	1983-10-25	...
Joyce	English	453453453	333445555	Joy	F	1958-05-03	...
Joyce	English	453453453	987654321	Abner	M	1942-02-28	...
Joyce	English	453453453	123456789	Michael	M	1988-01-04	...
Joyce	English	453453453	123456789	Alice	F	1988-12-30	...
Joyce	English	453453453	123456789	Elizabeth	F	1967-05-05	...

ACTUAL_DEPENDENTS

FNAME	LNAME	SSN	ESSN	DEPENDENT_NAME	SEX	BDATE
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28

RESULT

FNAME	LNAME	DEPENDENT_NAME
Jennifer	Wallace	Abner

Binary Relational Operations

- JOIN Operation
  - The sequence of cartesian product followed by select is used quite commonly to identify and select related tuples from two relations, a special operation, called JOIN. It is denoted by a ⋈
  - This operation is very important for any relational database with more than a single relation, because it allows us to process relationships among relations.
  - The general form of a join operation on two relations R(A1, A2, ..., An) and S(B1, B2, ..., Bm) is:
    - $R \bowtie_{\text{<join condition>}} S$
    - where R and S can be any relations that result from general relational algebra expressions.

Binary Relational Operations (cont.)

- Example: Suppose that we want to retrieve the name of the manager of each department. To get the manager's name, we need to combine each DEPARTMENT tuple with the EMPLOYEE tuple whose SSN value matches the MGRSSN value in the department tuple. We do this by using the join operation.

- DEPT_MGR ← DEPARTMENT $\bowtie_{\text{MGRSSN=SSN}}$ EMPLOYEE

DEPT_MGR	DNAME	DNUMBER	MGRSSN	. . .	FNAME	MINIT	LNAME	SSN	. . .
	Research	5	333445555	. . .	Franklin	T	Wong	333445555	. . .
	Administration	4	987654321	. . .	Jennifer	S	Wallace	987654321	. . .
	Headquarters	1	888665555	. . .	James	E	Borg	888665555	. . .

FIGURE 6.6 Result of the JOIN operation DEPT_MGR ← DEPARTMENT $\bowtie_{\text{MGRSSN=SSN}}$ EMPLOYEE.

Binary Relational Operations (cont.)

- **EQUIJOIN** Operation
- The most common use of join involves join conditions with equality comparisons only. Such a join, where the only comparison operator used is =, is called an EQUIJOIN. In the result of an EQUIJOIN we always have one or more pairs of attributes (whose names need not be identical) that have identical values in every tuple.
- The JOIN seen in the previous example was EQUIJOIN.

- **NATURAL JOIN** Operation
- Because one of each pair of attributes with identical values is superfluous, a new operation called natural join—denoted by * —was created to get rid of the second (superfluous) attribute in an EQUIJOIN condition.
- The standard definition of natural join requires that the two join attributes, or each pair of corresponding join attributes, have the same name in both relations. If this is not the case, a renaming operation is applied first.

Binary Relational Operations (cont.)

- Example: To apply a natural join on the DNUMBER attributes of DEPARTMENT and DEPT_LOCATIONS, it is sufficient to write:
  - DEPT_LOCS ← DEPARTMENT * DEPT_LOCATIONS

(a)

PROJ_DEPT	PNAME	PNUMBER	PLOCATION	DNUM	DNAME	MGRSSN	MGRSTARTDATE
	ProductX	1	Bellaire	5	Research	333445555	1988-05-22
	ProductY	2	Sugarland	5	Research	333445555	1988-05-22
	ProductZ	3	Houston	5	Research	333445555	1988-05-22
	Computerization	10	Stafford	4	Administration	987654321	1995-01-01
	Reorganization	20	Houston	1	Headquarters	888665555	1981-06-19
	Newbenefits	30	Stafford	4	Administration	987654321	1995-01-01

(b)

DEPT_LOCS	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE	LOCATION
	Headquarters	1	888665555	1981-06-19	Houston
	Administration	4	987654321	1995-01-01	Stafford
	Research	5	333445555	1988-05-22	Bellaire
	Research	5	333445555	1988-05-22	Sugarland
	Research	5	333445555	1988-05-22	Houston

FIGURE 6.7 Results of two NATURAL JOIN operations. (a) PROJ_DEPT ← PROJECT * DEPT. (b) DEPT_LOCS ← DEPARTMENT * DEPT_LOCATIONS.

Recap of Relational Algebra Operations

TABLE 6.1 OPERATIONS OF RELATIONAL ALGEBRA

Operation	Purpose	Notation
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{\text{<SELECTION CONDITION>}}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{\text{<ATTRIBUTE LIST>}}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{\text{<JOIN CONDITION>}} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{\text{<JOIN CONDITION>}} R_2$, OR $R_1 \bowtie (\text{<JOIN ATTRIBUTES 1>}), (\text{<JOIN ATTRIBUTES 2>})\ R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 *_{\text{<JOIN CONDITION>}} R_2$, OR $R_1 * (\text{<JOIN ATTRIBUTES 1>}), (\text{<JOIN ATTRIBUTES 2>}) R_2$ OR $R_1 * R_2$
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2.	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

Additional Relational Operations

- Aggregate Functions and Grouping
  - A type of request that cannot be expressed in the basic relational algebra is to specify mathematical aggregate functions on collections of values from the database.
  - Examples of such functions include retrieving the average or total salary of all employees or the total number of employee tuples. These functions are used in simple statistical queries that summarize information from the database tuples.
  - Common functions applied to collections of numeric values include SUM, AVERAGE, MAXIMUM, and MINIMUM. The COUNT function is used for counting tuples or values.

Additional Relational Operations (cont.)

(a)

R	DNO	NO_OF_EMPLOYEES	AVERAGE_SAL
	5	4	33250
	4	3	31000
	1	1	55000

(b)

DNO	COUNT_SSN	AVERAGE_SALARY
5	4	33250
4	3	31000
1	1	55000

(c)

COUNT_SSN	AVERAGE_SALARY
8	35125

Additional Relational Operations (cont.)

- Use of the Functional operator $\mathscr{F}$
  - $\mathscr{F}$MAX Salary (Employee) retrieves the maximum salary value from the Employee relation
  - $\mathscr{F}$MIN Salary (Employee) retrieves the minimum Salary value from the Employee relation
  - $\mathscr{F}$SUM Salary (Employee) retrieves the sum of the Salary from the Employee relation
  - DNO $\mathscr{F}$COUNT SSN, AVERAGE Salary (Employee) groups employees by DNO (department number) and computes the count of employees and average salary per department.[Note: count just counts the number of rows, without removing duplicates]

Additional Relational Operations (cont.)

- The OUTER JOIN Operation
  - In NATURAL JOIN tuples without a matching (or related) tuple are eliminated from the join result. Tuples with null in the join attributes are also eliminated. This amounts to loss of information.
  - A set of operations, called outer joins, can be used when we want to keep all the tuples in R, or all those in S, or all those in both relations in the result of the join, regardless of whether or not they have matching tuples in the other relation.
  - The left outer join operation keeps every tuple in the first or left relation R in R ⟕ S; if no matching tuple is found in S, then the attributes of S in the join result are filled or "padded" with null values.
  - A similar operation, right outer join, keeps every tuple in the second or right relation S in the result of R ⟖ S.
  - A third operation, full outer join, denoted by ⟗ keeps all tuples in both the left and the right relations when no matching tuples are found, padding them with null values as needed.

Additional Relational Operations (cont.)

RESULT	FNAME	MINIT	LNAME	DNAME
	John	B	Smith	null
	Franklin	T	Wong	Research
	Alicia	J	Zelaya	null
	Jennifer	S	Wallace	Administration
	Ramesh	K	Narayan	null
	Joyce	A	English	null
	Ahmad	V	Jabbar	null
	James	E	Borg	Headquarters

SQL

Data Definition, Constraints, and Schema Changes

- Used to CREATE, DROP, and ALTER the descriptions of the tables (relations) of a database

CREATE TABLE

- Specifies a new base relation by giving it a name, and specifying each of its attributes and their data types (INTEGER, FLOAT, DECIMAL(i,j), CHAR(n), VARCHAR(n))
- A constraint NOT NULL may be specified on an attribute

```
CREATE TABLE   DEPARTMENT
      (   DNAME   VARCHAR(10)   NOT NULL,
          DNUMBER   INTEGERNOT NULL,
          MGRSSN     CHAR(9),
          MGRSTARTDATE   CHAR(9)  );
```

CREATE TABLE

- In SQL2, can use the CREATE TABLE command for specifying the primary key attributes, secondary keys, and referential integrity constraints (foreign keys).
- Key attributes can be specified via the PRIMARY KEY and UNIQUE phrases
- CREATE TABLE DEPT
  (DNAME VARCHAR(10) NOT NULL,
  DNUMBER INTEGER NOT NULL,
  MGRSSN CHAR(9),
  MGRSTARTDATE CHAR(9),
  PRIMARY KEY (DNUMBER),
  UNIQUE (DNAME),
  FOREIGN KEY (MGRSSN) REFERENCES EMP);

DROP TABLE

- Used to remove a relation (base table) and its definition
- The relation can no longer be used in queries, updates, or any other commands since its description no longer exists
- Example:

  DROP TABLE DEPENDENT;

ALTER TABLE

- Used to add an attribute to one of the base relations
- The new attribute will have NULLs in all the tuples of the relation right after the command is executed; hence, the NOT NULL constraint is not allowed for such an attribute
- Example:

  ALTER TABLE EMPLOYEE ADD   JOB   VARCHAR(12);

- The database users must still enter a value for the new attribute JOB for each EMPLOYEE tuple. This can be done using the UPDATE command.

REFERENTIAL INTEGRITY OPTIONS

- We can specify RESTRICT, CASCADE, SET NULL or SET DEFAULT on referential integrity constraints (foreign keys)

```
CREATE TABLE   DEPT
 (     DNAME   VARCHAR(10)    NOT NULL,
       DNUMBER    INTEGER NOT NULL,
       MGRSSN CHAR(9),
       MGRSTARTDATE     CHAR(9),
       PRIMARY KEY (DNUMBER),
       UNIQUE (DNAME),
       FOREIGN KEY (MGRSSN) REFERENCES EMP
ON DELETE SET DEFAULT ON UPDATE CASCADE  );
```

REFERENTIAL INTEGRITY OPTIONS (continued)

```
CREATE TABLE   EMP
    (    ENAME    VARCHAR(30)     NOT NULL,
         ESSN  CHAR(9),
         BDATE     DATE,
         DNO   INTEGER  DEFAULT 1,
         SUPERSSN   CHAR(9),
         PRIMARY KEY (ESSN),
         FOREIGN KEY (DNO) REFERENCES DEPT
          ON DELETE SET DEFAULT ON UPDATE CASCADE,
         FOREIGN KEY (SUPERSSN) REFERENCES EMP
          ON DELETE SET NULL ON UPDATE CASCADE  );
```

Additional Data Types in SQL2 and SQL-99

- Has DATE, TIME, and TIMESTAMP data types
- DATE:
  - Made up of year-month-day in the format yyyy-mm-dd
- TIME:
  - Made up of hour:minute:second in the format hh:mm:ss
- TIME(i):
  - Made up of hour:minute:second plus i additional digits specifying fractions of a second
  - format is hh:mm:ss:ii...i
- TIMESTAMP:
  - Has both DATE and TIME components

Additional Data Types in SQL2 and SQL-99 (cont.)

- INTERVAL:
  - Specifies a relative value rather than an absolute value
  - Can be DAY/TIME intervals or YEAR/MONTH intervals
  - Can be positive or negative when added to or subtracted from an absolute value, the result is an absolute value

Retrieval Queries in SQL

- SQL has one basic statement for retrieving information from a database; the SELECT statement
- This is not the same as the SELECT operation of the relational algebra
- Important distinction between SQL and the formal relational model; SQL allows a table (relation) to have two or more tuples that are identical in all their attribute values
- Hence, an SQL relation (table) is a multi-set (sometimes called a bag) of tuples; it is not a set of tuples
- SQL relations can be constrained to be sets by specifying PRIMARY KEY or UNIQUE attributes, or by using the DISTINCT option in a query

Retrieval Queries in SQL (cont.)

- Basic form of the SQL SELECT statement is called a mapping or a SELECT-FROM-WHERE block

SELECT <attribute list>

FROM <table list>

WHERE <condition>

- <attribute list> is a list of attribute names whose values are to be retrieved by the query
- <table list> is a list of the relation names required to process the query
- <condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query

EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

DEPARTMENT

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE

DEPT_LOCATIONS

DNUMBER	DLOCATION

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM

WORKS_ON

ESSN	PNO	HOURS

DEPENDENT

ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP

EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	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	1968-07-19	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

DEPT_LOCATIONS	DNUMBER	DLOCATION
	1	Houston
	4	Stafford
	5	Bellaire
	5	Sugarland
	5	Houston

DEPARTMENT	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE
	Research	5	333445555	1988-05-22
	Administration	4	987654321	1995-01-01
	Headquarters	1	888665555	1981-06-19

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

Simple SQL Queries

- Basic SQL queries correspond to using the SELECT, PROJECT, and JOIN operations of the relational algebra
- All subsequent examples use the COMPANY database
- Example of a simple query on one relation
- Query 0: Retrieve the birthdate and address of the employee whose name is 'John B. Smith'.
  - Q0: SELECT BDATE, ADDRESS
    FROM EMPLOYEE
    WHERE FNAME='John' AND MINIT='B‘
    AND LNAME='Smith’
    - Similar to a SELECT-PROJECT pair of relational algebra operations; the SELECT-clause specifies the projection attributes and the WHERE-clause specifies the selection condition
    - However, the result of the query may contain duplicate tuples

Simple SQL Queries (cont.)

- Query 1: Retrieve the name and address of all employees who work for the 'Research' department.

  - Q1: SELECT FNAME, LNAME, ADDRESS
    FROM EMPLOYEE, DEPARTMENT
    WHERE DNAME='Research' AND DNUMBER=DNO

  - Similar to a SELECT-PROJECT-JOIN sequence of relational algebra operations
  - (DNAME='Research') is a selection condition (corresponds to a SELECT operation in relational algebra)
  - (DNUMBER=DNO) is a join condition (corresponds to a JOIN operation in relational algebra)

Simple SQL Queries (cont.)

- Query 2: For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birthdate.

  - Q2: SELECT PNUMBER, DNUM, LNAME, BDATE, ADDRESS
    FROM PROJECT, DEPARTMENT, EMPLOYEE
    WHERE DNUM=DNUMBER AND MGRSSN=SSN
    AND PLOCATION='Stafford'

- In Q2, there are two join conditions
- The join condition DNUM=DNUMBER relates a project to its controlling department
- The join condition MGRSSN=SSN relates the controlling department to the employee who manages that department

Aliases, * and DISTINCT, Empty WHERE-clause

- In SQL, we can use the same name for two (or more) attributes as long as the attributes are in different relations
  A query that refers to two or more attributes with the same name must qualify the attribute name with the relation name by prefixing the relation name to the attribute name

- Example:

  - EMPLOYEE.LNAME, DEPARTMENT.DNAME

ALIASES

- Some queries need to refer to the same relation twice
- In this case, aliases are given to the relation name
- Query 8: For each employee, retrieve the employee's name, and the name of his or her immediate supervisor.

```
Q8:   SELECT  E.FNAME, E.LNAME, S.FNAME, S.LNAME
      FROM    EMPLOYEE E S
      WHERE   E.SUPERSSN=S.SSN
```

- In Q8, the alternate relation names E and S are called aliases or tuple variables for the EMPLOYEE relation
- We can think of E and S as two different copies of EMPLOYEE; E represents employees in role of supervisees and S represents employees in role of supervisors

ALIASES (cont.)

- Aliasing can also be used in any SQL query for convenience
  Can also use the AS keyword to specify aliases

```
Q8:   SELECT   E.FNAME, E.LNAME, S.FNAME, S.LNAME
      FROM EMPLOYEE AS E, EMPLOYEE AS S
      WHERE    E.SUPERSSN=S.SSN
```

UNSPECIFIED WHERE-clause

- A missing WHERE-clause indicates no condition; hence, all tuples of the relations in the FROM-clause are selected
- This is equivalent to the condition WHERE TRUE
- Query 9: Retrieve the SSN values for all employees.
  - Q9: SELECT SSN
    FROM EMPLOYEE
- If more than one relation is specified in the FROM-clause and there is no join condition, then the CARTESIAN PRODUCT of tuples is selected

UNSPECIFIED WHERE-clause (cont.)

- Example:

```
Q10:   SELECT SSN, DNAME
       FROM   EMPLOYEE, DEPARTMENT
```

- It is extremely important not to overlook specifying any selection and join conditions in the WHERE-clause; otherwise, incorrect and very large relations may result

USE OF *

- To retrieve all the attribute values of the selected tuples, a * is used, which stands for all the attributes
  Examples:

- Q1C:
```
SELECT  *
FROM EMPLOYEE
WHERE  DNO=5
```

Q1D:
```
SELECT  *
FROM EMPLOYEE, DEPARTMENT
WHERE  DNAME='Research' AND
       DNO=DNUMBER
```

USE OF DISTINCT

- SQL does not treat a relation as a set; duplicate tuples can appear
- To eliminate duplicate tuples in a query result, the keyword DISTINCT is used
- For example, the result of Q11 may have duplicate SALARY values whereas Q11A does not have any duplicate values

- Q11: SELECT SALARY
  FROM EMPLOYEE
  Q11A: SELECT DISTINCT SALARY
  FROM EMPLOYEE

SET OPERATIONS

- SQL has directly incorporated some set operations
- There is a **union** operation (UNION), and in some versions of SQL there are set difference (MINUS) and intersection (INTERSECT) operations
- The resulting relations of these set operations are sets of tuples; duplicate tuples are eliminated from the result
- The set operations apply only to union compatible relations; the two relations must have the same attributes and the attributes must appear in the same order

SET OPERATIONS (cont.)

- Query 4: Make a list of all project numbers for projects that involve an employee whose last name is 'Smith' as a worker or as a manager of the department that controls the project.

```
Q4:  (SELECT  PNAME
      FROM    PROJECT, DEPARTMENT, EMPLOYEE
      WHERE   DNUM=DNUMBER AND MGRSSN=SSN
              AND LNAME='Smith')
      UNION
      (SELECT  PNAME
      FROM    PROJECT, WORKS_ON, EMPLOYEE
      WHERE   PNUMBER=PNO AND ESSN=SSN
              AND LNAME='Smith')
```

NESTING OF QUERIES

- A complete SELECT query, called a nested query, can be specified within the WHERE-clause of another query, called the outer query
- Many of the previous queries can be specified in an alternative form using nesting

- Query 1: Retrieve the name and address of all employees who work for the 'Research' department.

```
Q1:  SELECT   FNAME, LNAME, ADDRESS
     FROM     EMPLOYEE
     WHERE    DNO IN  (SELECT  DNUMBER
     FROM     DEPARTMENT
     WHERE    DNAME='Research' )
```

NESTING OF QUERIES (cont.)

- The nested query selects the number of the 'Research' department
- The outer query select an EMPLOYEE tuple if its DNO value is in the result of either nested query
- The comparison operator IN compares a value v with a set (or multi-set) of values V, and evaluates to TRUE if v is one of the elements in V
- In general, we can have several levels of nested queries
- A reference to an unqualified attribute refers to the relation declared in the innermost nested query
- In this example, the nested query is not correlated with the outer query

CORRELATED NESTED QUERIES

- If a condition in the WHERE-clause of a nested query references an attribute of a relation declared in the outer query, the two queries are said to be correlated
- The result of a correlated nested query is different for each tuple (or combination of tuples) of the relation(s) in the outer query
- Query 12: Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
Q12: SELECT     E.FNAME, E.LNAME
     FROM       EMPLOYEE AS E
     WHERE      E.SSN IN    (SELECT ESSN
                             FROM   DEPENDENT
                             WHERE  ESSN=E.SSN AND
                                    E.FNAME=DEPENDENT_NAME)
```

CORRELATED NESTED QUERIES (cont.)

- In Q12, the nested query has a different result for each tuple in the outer query
- A query written with nested SELECT... FROM... WHERE... blocks and using the = or IN comparison operators can always be expressed as a single block query. For example, Q12 may be written as in Q12A

```
Q12A:    SELECT   E.FNAME, E.LNAME
         FROM     EMPLOYEE E, DEPENDENT D
         WHERE        E.SSN=D.ESSN AND
                      E.FNAME=D.DEPENDENT_NAME
```

THE EXISTS FUNCTION

- EXISTS is used to check whether the result of a correlated nested query is empty (contains no tuples) or not
- We can formulate Query 12 in an alternative form that uses EXISTS as Q12B below
- Query 12: Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
Q12B:   SELECT FNAME, LNAME
        FROM   EMPLOYEE
        WHERE EXISTS     (SELECT    *
                          FROM       DEPENDENT
                          WHERE      SSN=ESSN AND
                                     FNAME=DEPENDENT_NAME)
```

THE EXISTS FUNCTION (cont.)

- Query 6: Retrieve the names of employees who have no dependents.

```
Q6:     SELECT   FNAME, LNAME
        FROM     EMPLOYEE
        WHERE    NOT EXISTS   (SELECT     *
                              FROM  DEPENDENT
                              WHERE SSN=ESSN)
```

- In Q6, the correlated nested query retrieves all DEPENDENT tuples related to an EMPLOYEE tuple. If none exist , the EMPLOYEE tuple is selected
- EXISTS is necessary for the expressive power of SQL

EXPLICIT SETS

- It is also possible to use an explicit (enumerated) set of values in the WHERE-clause rather than a nested query

- Query 13: Retrieve the social security numbers of all employees who work on project number 1, 2, or 3.

  - Q13: SELECT DISTINCT ESSN
    FROM WORKS_ON
    WHERE PNO IN (1, 2, 3)

NULLS IN SQL QUERIES

- SQL allows queries that check if a value is NULL (missing or undefined or not applicable)
- SQL uses IS or IS NOT to compare NULLs because it considers each NULL value distinct from other NULL values, so equality comparison is not appropriate .
- Query 14: Retrieve the names of all employees who do not have supervisors.

```
Q14: SELECT     FNAME, LNAME
     FROM       EMPLOYEE
     WHERESUPERSSN  IS  NULL
```

Note: If a join condition is specified, tuples with NULL values for the join attributes are not included in the result

Joined Relations Feature in SQL2

- Can specify a "joined relation" in the FROM-clause
- Looks like any other relation but is the result of a join
- Allows the user to specify different types of joins (regular "theta" JOIN, NATURAL JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, CROSS JOIN, etc)

Joined Relations Feature in SQL2 (cont.)

- Examples:

```
Q8:   SELECT     E.FNAME, E.LNAME, S.FNAME, S.LNAME
      FROM       EMPLOYEE E S
      WHERE      E.SUPERSSN=S.SSN
```

can be written as:

```
Q8:   SELECT     E.FNAME, E.LNAME, S.FNAME, S.LNAME
      FROM       (EMPLOYEE E LEFT OUTER JOIN EMPLOYEES
                      ON  E.SUPERSSN=S.SSN)
```

Joined Relations Feature in SQL2 (cont.)

- Q1: SELECT FNAME, LNAME, ADDRESS
  FROM EMPLOYEE, DEPARTMENT
  WHERE DNAME='Research' AND DNUMBER=DNO

  could be written as:

  Q1: SELECT FNAME, LNAME, ADDRESS
  FROM (EMPLOYEE JOIN DEPARTMENT
  ON DNUMBER=DNO)
  WHERE DNAME='Research'

  or as:

  Q1: SELECT FNAME, LNAME, ADDRESS
  FROM (EMPLOYEE NATURAL JOIN DEPARTMENT
  AS DEPT(DNAME, DNO, MSSN, MSDATE)
  WHERE DNAME='Research'

Joined Relations Feature in SQL2 (cont.)

- Another Example;
  - Q2 could be written as follows; this illustrates multiple joins in the joined tables

```
Q2:   SELECT PNUMBER, DNUM, LNAME, BDATE, ADDRESS
      FROM   (PROJECT JOIN DEPARTMENT ON DNUM=DNUMBER)
                    JOIN EMPLOYEE ON MGRSSN=SSN
      WHERE  PLOCATION='Stafford’
```

AGGREGATE FUNCTIONS

- Include COUNT, SUM, MAX, MIN, and AVG
- Query 15: Find the maximum salary, the minimum salary, and the average salary among all employees.

```
Q15: SELECT   MAX(SALARY), MIN(SALARY), AVG(SALARY)
     FROM     EMPLOYEE
```

- Some SQL implementations may not allow more than one function in the SELECT-clause

AGGREGATE FUNCTIONS (cont.)

- Query 16: Find the maximum salary, the minimum salary, and the average salary among employees who work for the 'Research' department.

```
Q16: SELECT  MAX(SALARY), MIN(SALARY), AVG(SALARY)
     FROM    EMPLOYEE, DEPARTMENT
     WHERE   DNO=DNUMBER AND DNAME='Research'
```

AGGREGATE FUNCTIONS (cont.)

- Queries 17 and 18: Retrieve the total number of employees in the company (Q17), and the number of employees in the 'Research' department (Q18).

```
Q17:  SELECT    COUNT (*)
      FROM      EMPLOYEE

Q18:  SELECT    COUNT (*)
      FROM      EMPLOYEE, DEPARTMENT
      WHERE     DNO=DNUMBER AND
                DNAME='Research'
```

GROUPING

- In many cases, we want to apply the aggregate functions to subgroups of tuples in a relation
- Each subgroup of tuples consists of the set of tuples that have the same value for the grouping attribute(s)
- The function is applied to each subgroup independently
- SQL has a GROUP BY-clause for specifying the grouping attributes, which must also appear in the SELECT-clause

GROUPING (cont.)

- Query 20: For each department, retrieve the department number, the number of employees in the department, and their average salary.

```
Q20:  SELECT      DNO, COUNT (*), AVG (SALARY)
      FROM        EMPLOYEE
      GROUP BY    DNO
```

- In Q20, the EMPLOYEE tuples are divided into groups--each group having the same value for the grouping attribute DNO
- The COUNT and AVG functions are applied to each such group of tuples separately
- The SELECT-clause includes only the grouping attribute and the functions to be applied on each group of tuples
- A join condition can be used in conjunction with grouping

GROUPING (cont.)

- Query 21: For each project, retrieve the project number, project name, and the number of employees who work on that project.

```
Q21:    SELECT      PNUMBER, PNAME, COUNT (*)
        FROM        PROJECT, WORKS_ON
        WHERE       PNUMBER=PNO
        GROUP BY    PNUMBER, PNAME
```

- In this case, the grouping and functions are applied after the joining of the two relations

THE HAVING-CLAUSE

- Sometimes we want to retrieve the values of these functions for only those groups that satisfy certain conditions
- The HAVING-clause is used for specifying a selection condition on groups (rather than on individual tuples)

THE HAVING-CLAUSE (cont.)

- Query 22: For each project on which more than two employees work, retrieve the project number, project name, and the number of employees who work on that project.

```
Q22:    SELECT      PNUMBER, PNAME, COUNT(*)
        FROM        PROJECT, WORKS_ON
        WHERE       PNUMBER=PNO
        GROUP BY    PNUMBER, PNAME
        HAVING      COUNT (*) > 2
```

SUBSTRING COMPARISON

- The LIKE comparison operator is used to compare partial strings
- Two reserved characters are used: '%' (or '*' in some implementations) replaces an arbitrary number of characters, and '_' replaces a single arbitrary character

SUBSTRING COMPARISON (cont.)

- Query 25: Retrieve all employees whose address is in Houston, Texas. Here, the value of the ADDRESS attribute must contain the substring 'Houston,TX'.

```
Q25:  SELECT FNAME, LNAME
      FROM   EMPLOYEE
      WHERE  ADDRESS LIKE '%Houston,TX%’
```

SUBSTRING COMPARISON (cont.)

- Query 26: Retrieve all employees who were born during the 1950s. Here, '5' must be the 8th character of the string (according to our format for date), so the BDATE value is '_____5_', with each underscore as a place holder for a single arbitrary character.

```
Q26:   SELECT FNAME, LNAME
       FROM   EMPLOYEE
       WHERE  BDATE LIKE   '_____5_'
```

ARITHMETIC OPERATIONS

- The standard arithmetic operators '+', '-'. '*', and '/' (for addition, subtraction, multiplication, and division, respectively) can be applied to numeric values in an SQL query result

- Query 27: Show the effect of giving all employees who work on the 'ProductX' project a 10% raise.

```
Q27:    SELECT  FNAME, LNAME, 1.1*SALARY
        FROM    EMPLOYEE, WORKS_ON, PROJECT
        WHERE   SSN=ESSN AND PNO=PNUMBER
                AND PNAME='ProductX’
```

ORDER BY

- The ORDER BY clause is used to sort the tuples in a query result based on the values of some attribute(s)
- Query 28: Retrieve a list of employees and the projects each works in, ordered by the employee's department, and within each department ordered alphabetically by employee last name.

```
Q28:    SELECT      DNAME, LNAME, FNAME, PNAME
        FROM        DEPARTMENT, EMPLOYEE,  WORKS_ON,
                    PROJECT
        WHERE       DNUMBER=DNO AND SSN=ESSN
                    AND PNO=PNUMBER
        ORDER BY    DNAME, LNAME
```

ORDER BY (cont.)

- The default order is in ascending order of values
- We can specify the keyword DESC if we want a descending order; the keyword ASC can be used to explicitly specify ascending order, even though it is the default

Summary of SQL Queries

- A query in SQL can consist of up to six clauses, but only the first two, SELECT and FROM, are mandatory. The clauses are specified in the following order:

```
SELECT      <attribute list>
FROM        <table list>
[WHERE      <condition>]
[GROUP BY   <grouping attribute(s)>]
[HAVING     <group condition>]
[ORDER BY   <attribute list>]
```

Summary of SQL Queries (cont.)

- The SELECT-clause lists the attributes or functions to be retrieved
- The FROM-clause specifies all relations (or aliases) needed in the query but not those needed in nested queries
- The WHERE-clause specifies the conditions for selection and join of tuples from the relations specified in the FROM-clause
- GROUP BY specifies grouping attributes
- HAVING specifies a condition for selection of groups
- ORDER BY specifies an order for displaying the result of a query
- A query is evaluated by first applying the WHERE-clause, then GROUP BY and HAVING, and finally the SELECT-clause

Specifying Updates in SQL

- There are three SQL commands to modify the database; INSERT, DELETE, and UPDATE

INSERT

- In its simplest form, it is used to add one or more tuples to a relation
- Attribute values should be listed in the same order as the attributes were specified in the CREATE TABLE command

INSERT (cont.)

- Example:

  U1: INSERT INTO EMPLOYEE
  VALUES ('Richard','K','Marini', '653298653', '30-DEC-52',
  '98 Oak Forest,Katy,TX', 'M', 37000,'987654321', 4)

- An alternate form of INSERT specifies explicitly the attribute names that correspond to the values in the new tuple
- Attributes with NULL values can be left out
- Example: Insert a tuple for a new EMPLOYEE for whom we only know the FNAME, LNAME, and SSN attributes.

  U1A: INSERT INTO EMPLOYEE (FNAME, LNAME, SSN)
  VALUES ('Richard', 'Marini', '653298653')

INSERT (cont.)

- Important Note: Only the constraints specified in the DDL commands are automatically enforced by the DBMS when updates are applied to the database

- Another variation of INSERT allows insertion of multiple tuples resulting from a query into a relation

INSERT (cont.)

- Example: Suppose we want to create a temporary table that has the name, number of employees, and total salaries for each department. A table DEPTS_INFO is created by U3A, and is loaded with the summary information retrieved from the database by the query in U3B.

```
U3A:    CREATE TABLE  DEPTS_INFO (
            DEPT_NAME       VARCHAR(10),
            NO_OF_EMPS      INTEGER,
            TOTAL_SAL       INTEGER);

U3B:    INSERT INTO DEPTS_INFO (DEPT_NAME, NO_OF_EMPS, TOTAL_SAL)
        SELECT      DNAME, COUNT (*), SUM (SALARY)
        FROM        DEPARTMENT, EMPLOYEE
        WHERE       DNUMBER=DNO
        GROUP BY    DNAME ;
```

INSERT (cont.)

- Note: The DEPTS_INFO table may not be up-to-date if we change the tuples in either the DEPARTMENT or the EMPLOYEE relations after issuing U3B. We have to create a view (see later) to keep such a table up to date.

DELETE

- Removes tuples from a relation
- Includes a WHERE-clause to select the tuples to be deleted
- Tuples are deleted from only one table at a time (unless CASCADE is specified on a referential integrity constraint)
- A missing WHERE-clause specifies that all tuples in the relation are to be deleted; the table then becomes an empty table
- The number of tuples deleted depends on the number of tuples in the relation that satisfy the WHERE-clause
- Referential integrity should be enforced

DELETE (cont.)

- Examples:

```
U4A: DELETE FROM    EMPLOYEE
       WHERE        LNAME='Brown’

U4B: DELETE FROM    EMPLOYEE
       WHERE        SSN='123456789’

U4C: DELETE FROM    EMPLOYEE
       WHERE        DNO  IN (SELECT    DNUMBER
                             FROM      DEPARTMENT
                             WHERE     DNAME='Research')

U4D: DELETE FROM    EMPLOYEE
```

UPDATE

- Used to modify attribute values of one or more selected tuples
- A WHERE-clause selects the tuples to be modified
- An additional SET-clause specifies the attributes to be modified and their new values
- Each command modifies tuples in the same relation
- Referential integrity should be enforced

UPDATE (cont.)

- Example: Change the location and controlling department number of project number 10 to 'Bellaire' and 5, respectively.

```
U5: UPDATE    PROJECT
    SET       PLOCATION = 'Bellaire', DNUM = 5
    WHERE     PNUMBER=10
```

UPDATE (cont.)

- Example: Give all employees in the 'Research' department a 10% raise in salary.

```
U6:   UPDATE EMPLOYEE
      SET    SALARY = SALARY *1.1
      WHERE  DNO  IN (SELECT DNUMBER
                      FROM   DEPARTMENT
                      WHERE  DNAME='Research')
```

- In this request, the modified SALARY value depends on the original SALARY value in each tuple
- The reference to the SALARY attribute on the right of = refers to the old SALARY value before modification
- The reference to the SALARY attribute on the left of = refers to the new SALARY value after modification

DATABASE IN PRACTICE

Examples

- A suite for distributed execution of general purpose applications on the Grid for small- and mid-size VOs
- Usage and performance analysis of Dirac databases (just a few comments)

Submission system

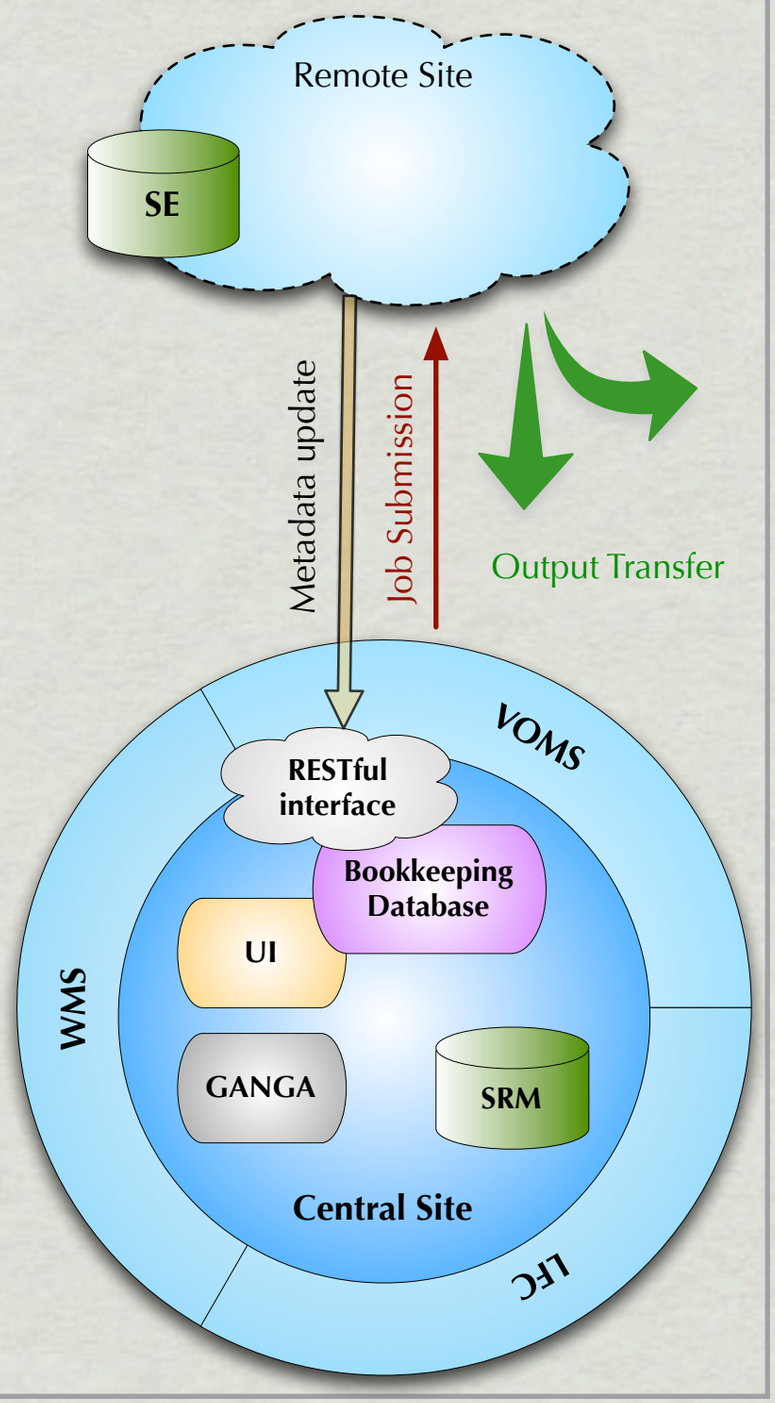
Aim

- Develop a flexible and dynamic model to manage (simulation) data production on the Grid for small- and medium VOs
- Allow a easy, quick and customizable access to the Grid Infrastructure to research groups/organizations
- Successfully tested with SuperB Full- and Fast Simulation Production

Distributed Architecture and Infrastructure

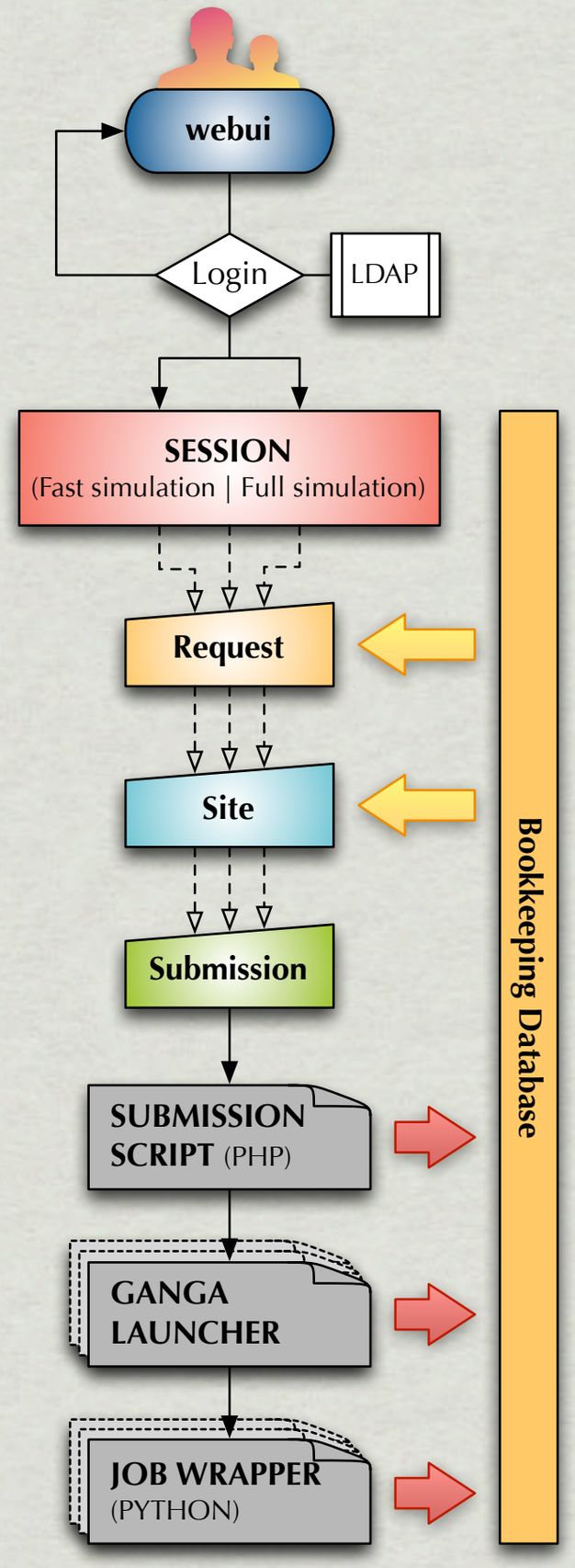
- The LHC Computing Grid (LCG) architecture was adopted to provide the minimum set of services and applications upon which the distributed production system has been built.

- Authentication and authorization is provided by VOMS service, LFC is the file catalog, WMS is used for brokering purpose and for Grid flavor interoperability features, transfers are done via Lcg-Utility, GANGA is the submitting interface.



System Design

- The simulation production system heavily relies on a bookkeeping database, storing both application-specific and infrastructure metadata, which is tightly coupled with a Web-based user-interface (WebUI).
- The first makes available to the users information on the execution status of jobs and their specific meaning and parameters, and contributes in orchestrating the submission mechanism.
- The latter provides job submission management for several simulation applications, bookkeeping database interactions and basic monitoring functionalities.



System Design

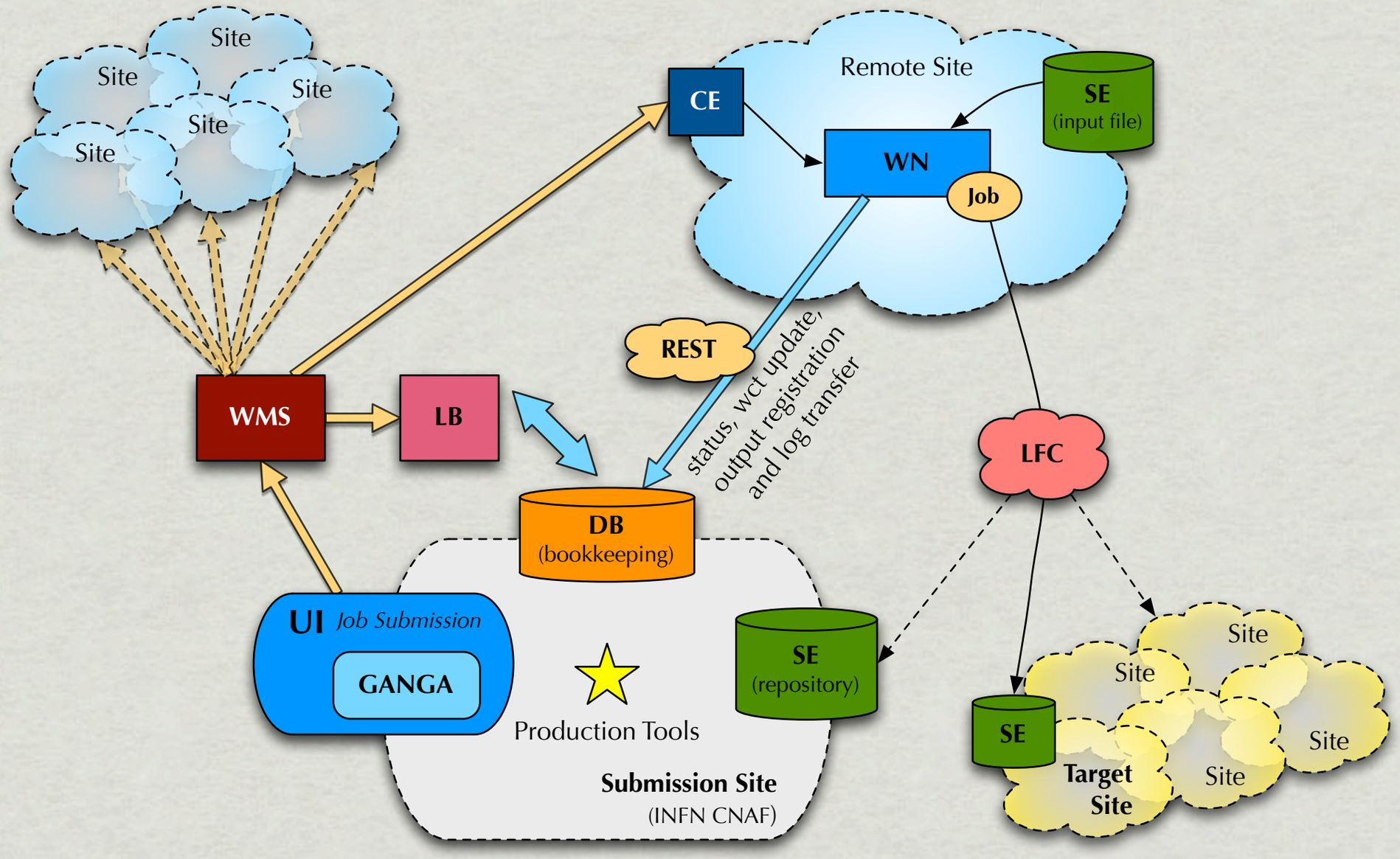
- The bookkeeping database is implemented with PostgreSQL rDBMS in a centralized way, the WebUI in PHP and JQuery.

- The database interactions with the submission portal and the job in execution on the WNs are managed by a direct interface to PostgreSQL or a RESTful interface (with X509 proxy-certificate cipher-encryption auths), respectively.

Job Workflow

- The structure of services and job workflow follow a semi-centralized design: job management service, bookkeeping database and default storage repository are hosted in a central site.
- Jobs executed into remote sites update the bookkeeping database with status, logging and timing information and transfer their output back to central repository or to a predefined site, discriminating on execution metadata.
- The system requires a proper configuration of the remote Grid sites.
- Bookkeeping metadata are integrated with Grid Logging & Bookkeeping service (LB) information provided by the infrastructure.
- In addition, the submission mechanism takes into account sites availability data from Nagios monitoring service.
- Simulation jobs are also exposed to the Grid dashboard for monitoring.

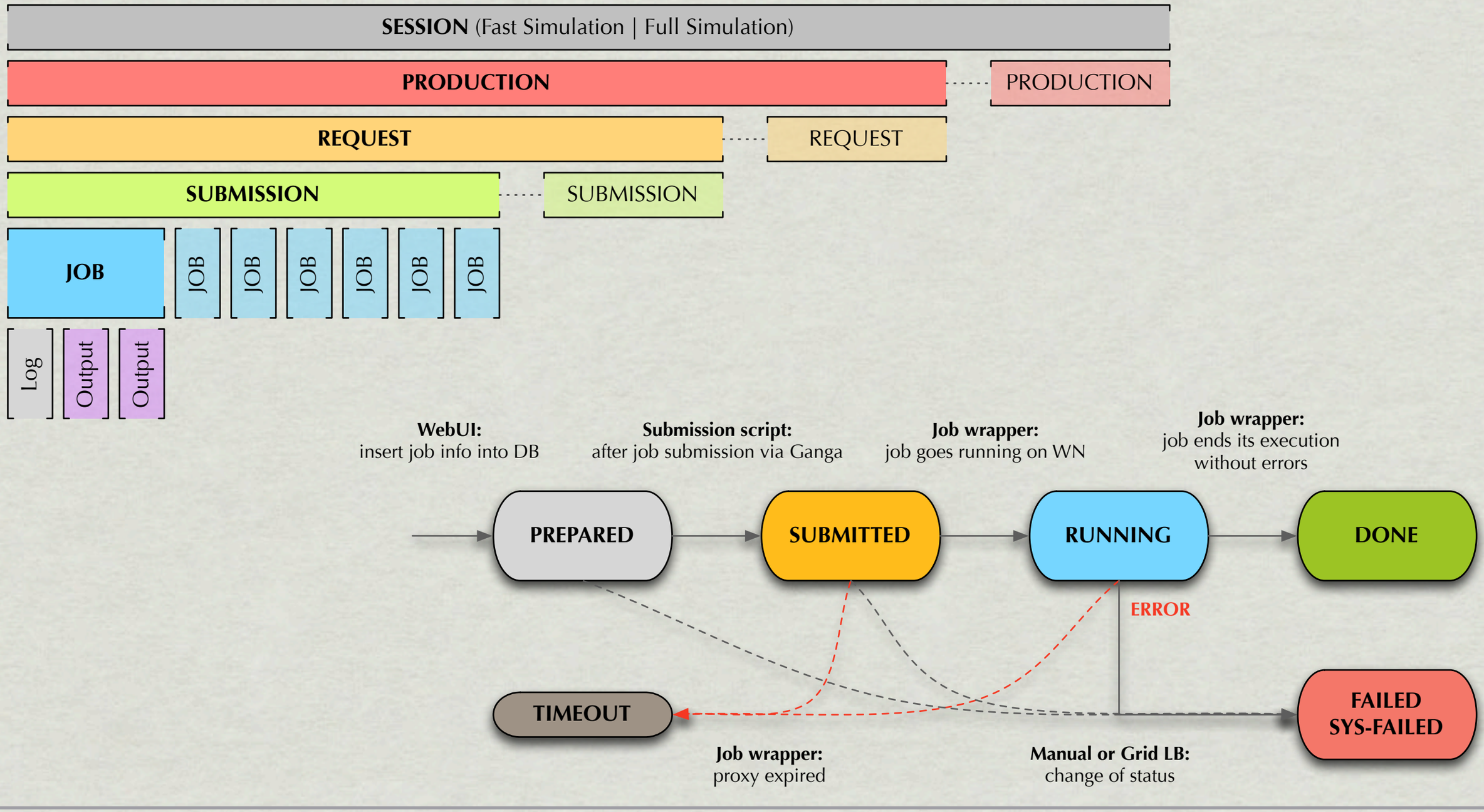
Job Workflow



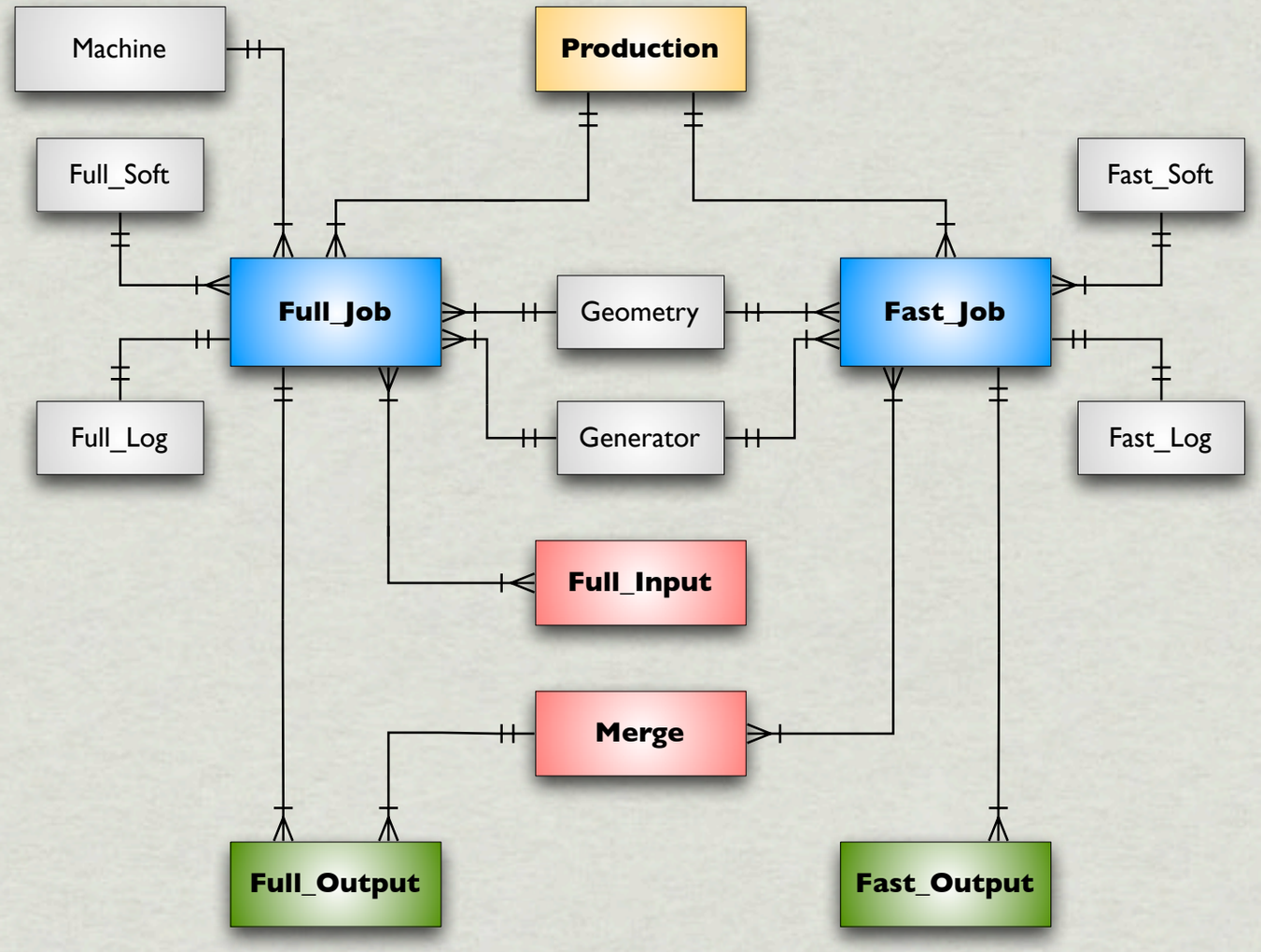
WebUI

- The WebUI provides separate management for several type of simulation productions. Both sections are divided in configuration, submission and a monitor subsections. Their content is dynamically generated from the bookkeeping database schema and state in order to include the simulation-specific fields.
- A production cycle consists of several requests (defined by a specific set of job parameters values and events), which in turn are divided in several submissions, each consisting of several jobs.
- A configuration interface for requests definition per production cycle and simulation type, is provided
- Multi-site submissions based on requests and fine grain parametric submission interfaces complete the set of available services permitting a shift based scheduled session and a debugging specific console, respectively.

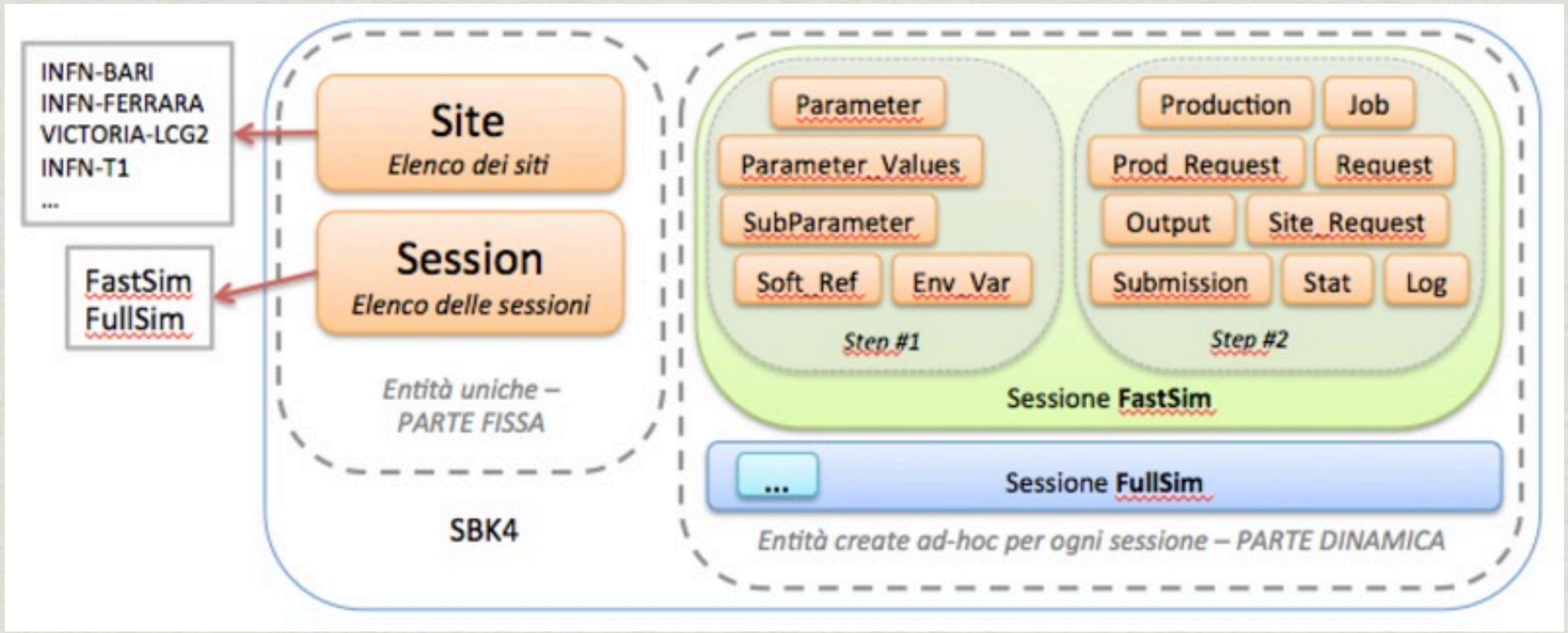
Sessions and status



Database schema
(simplified ER/Relational schema)



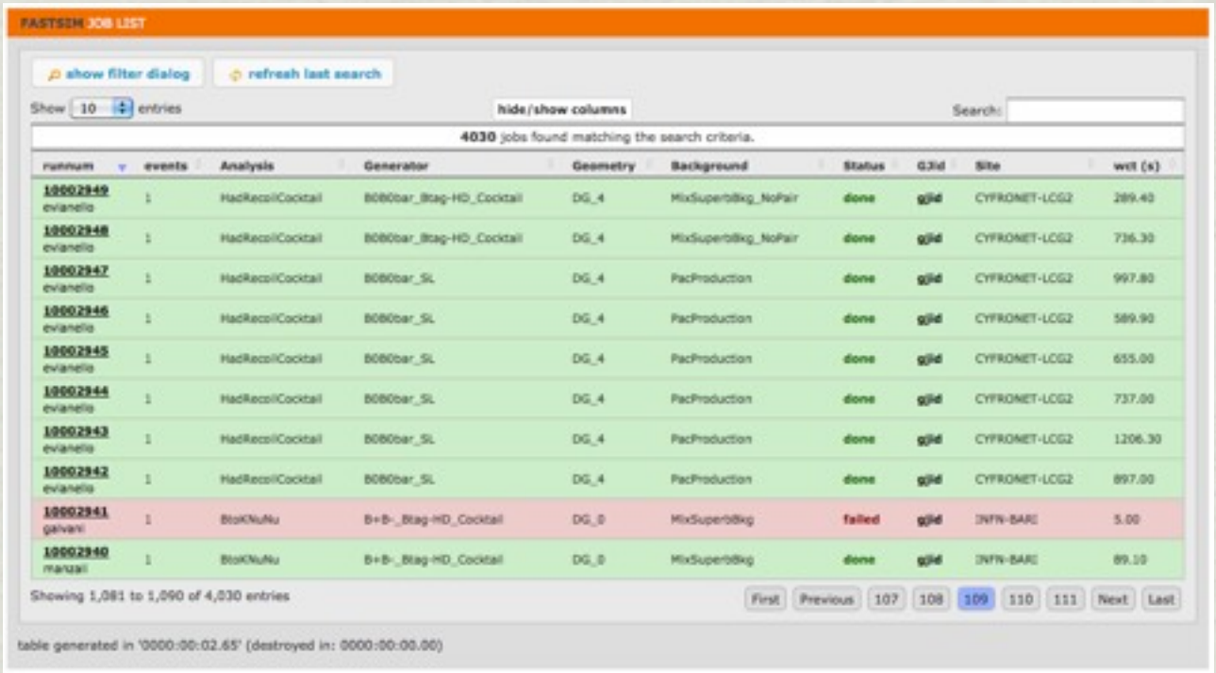
Database structure



in use...


Home Page
USER INFO:
Hi Enrico Vianello!
user-id: evianello
auto-sub: enabled
you are member of 5 group(s):
- webui
- webuiFastProdMgr
- webuiFullProdMgr
- webuiShift
- WebUiDeveloper
user privileges for FastSim:
create productions
create requests
shifter ui
expert ui
monitoring jobs
user privileges for FullSim:
create productions
create requests
shifter ui
expert ui
monitoring jobs
GUIDE
RESOURCES
YOUR SESSIONS' STATISTICS:
session | job launched | job graph | events generated | total wct (s)
FastSim 4|09 | 1454079 | 44|2912.80
FullSim 253 | 253 | 120790.90
YOUR LAST 10 ACTIVITIES:
Show 10 entries
Search:
time | type | session | info
2011-11-27 22:26:15 JOBSUBMISSION FastSim 6 jobs (6 events) on 2010_July
2011-11-24 11:52:49 JOBSUBMISSION FastSim 10 jobs (300000 events) on 2010_July_test
2011-11-22 18:02:40 JOBSUBMISSION FastSim 3 jobs (3 events) on 2010_July_test
2011-11-22 16:35:15 JOBSUBMISSION FullSim 7 jobs (7 events) on 2011_Full_grid_test_test
2011-11-22 12:01:11 JOBSUBMISSION FastSim 15 jobs (750000 events) on 2010_July_test
2011-11-18 09:32:01 JOBSUBMISSION FastSim 50 jobs (50 events) on 2010_July
2011-11-16 12:27:21 JOBSUBMISSION FullSim 1 jobs (1 events) on 2011_Full_grid_test_test
2011-11-16 12:24:23 JOBSUBMISSION FastSim 1 jobs (1 events) on 2010_July_test
2011-11-16 11:52:31 JOBSUBMISSION FullSim 1 jobs (1 events) on 2011_Full_grid_test_test
2011-11-15 13:53:23 JOBSUBMISSION FullSim 2 jobs (2 events) on 2011_Full_grid_test_test
Showing 1 to 10 of 10 entries
First Previous 1 Next Last


FASTSIM PRODUCTION REQUESTS
List of [2] 2010_July requests
show test requests
Show 10 entries
hide/show columns
Search:
Analysis | Generator | Geometry | Background | req.events | evt/job | #jobs | minrunnum | status | vid | operations
HadRecoilCocktail | BOBObar_SL | DG_4 | PacProduction | 37600 | 1 | 37600 | 100000000 | running | evianello
progress: none
events: 2381/37600
Jobs per status:
Events per status:
Jobs' status per site:
done
failed
Last submission:
at 2011-11-27 22:26:15 on INFN-PERUGIA
by evianello
total jobs: 6
total events: 6
total wct: 0.00
view details
HadRecoilCocktail | BOBObar_Btag-HD_Cocktail | DG_4 | MixSuperbBkg_NoPair | 3760000000 | 80000 | 4700 | 100000000 | running | evianello
Showing 1 to 2 of 2 entries
First Previous 1 Next Last


FASTSIM JOB LIST
show filter dialog
refresh last search
Show 10 entries
hide/show columns
Search:
4030 jobs found matching the search criteria.
runnum | events | Analysis | Generator | Geometry | Background | Status | GJid | Site | wct (s)
10002949 evianello | 1 | HadRecoilCocktail | BOBObar_Btag-HD_Cocktail | DG_4 | MixSuperbBkg_NoPair | done | gjid | CYFRONET-LCG2 | 289.40
10002948 evianello | 1 | HadRecoilCocktail | BOBObar_Btag-HD_Cocktail | DG_4 | MixSuperbBkg_NoPair | done | gjid | CYFRONET-LCG2 | 736.30
10002947 evianello | 1 | HadRecoilCocktail | BOBObar_SL | DG_4 | PacProduction | done | gjid | CYFRONET-LCG2 | 997.80
10002946 evianello | 1 | HadRecoilCocktail | BOBObar_SL | DG_4 | PacProduction | done | gjid | CYFRONET-LCG2 | 589.90
10002945 evianello | 1 | HadRecoilCocktail | BOBObar_SL | DG_4 | PacProduction | done | gjid | CYFRONET-LCG2 | 655.00
10002944 evianello | 1 | HadRecoilCocktail | BOBObar_SL | DG_4 | PacProduction | done | gjid | CYFRONET-LCG2 | 737.00
10002943 evianello | 1 | HadRecoilCocktail | BOBObar_SL | DG_4 | PacProduction | done | gjid | CYFRONET-LCG2 | 1206.30
10002942 evianello | 1 | HadRecoilCocktail | BOBObar_SL | DG_4 | PacProduction | done | gjid | CYFRONET-LCG2 | 897.00
10002941 galvani | 1 | BtoKNuNu | B+B-_Btag-HD_Cocktail | DG_0 | MixSuperbBkg | failed | gjid | INFN-BARI | 5.00
10002940 manzali | 1 | BtoKNuNu | B+B-_Btag-HD_Cocktail | DG_0 | MixSuperbBkg | done | gjid | INFN-BARI | 89.10
Showing 1,081 to 1,090 of 4,030 entries
First Previous 107 108 109 110 111 Next Last
table generated in '0000:00:02.65' (destroyed in: 0000:00:00.00)

in use...

Productions [FastSim]

FASTSIM PRODUCTION LIST

List of session's productions

☑ show test productions

Show 10 entries Search:

Prod Series	Status	Soft Release	Job Stats	Requests	actions	note
2010_July		V0.2.5 311		**2** Requests (**0** open, **0** valid, **2** running, **0** closed)	close	
2010_July_test		V0.2.5 311		**7** Requests		
2010_September_307		V0.2.5 307		**56** Requests (**0** open, **0** valid, **0** running, **56** closed)	open	
2010_September_307_test		V0.2.5 307		**2** Requests		
2010_September_311		V0.2.5 311		**43** Requests (**0** open, **0** valid, **0** running, **43** closed)	open	
2010_September_311_test		V0.2.5 311		**18** Requests		

Showing 1 to 6 of 6 entries

First Previous 1 Next Last

FASTSIM CREATE PRODUCTION:

All form fields are required.

Production Series: 2011_November

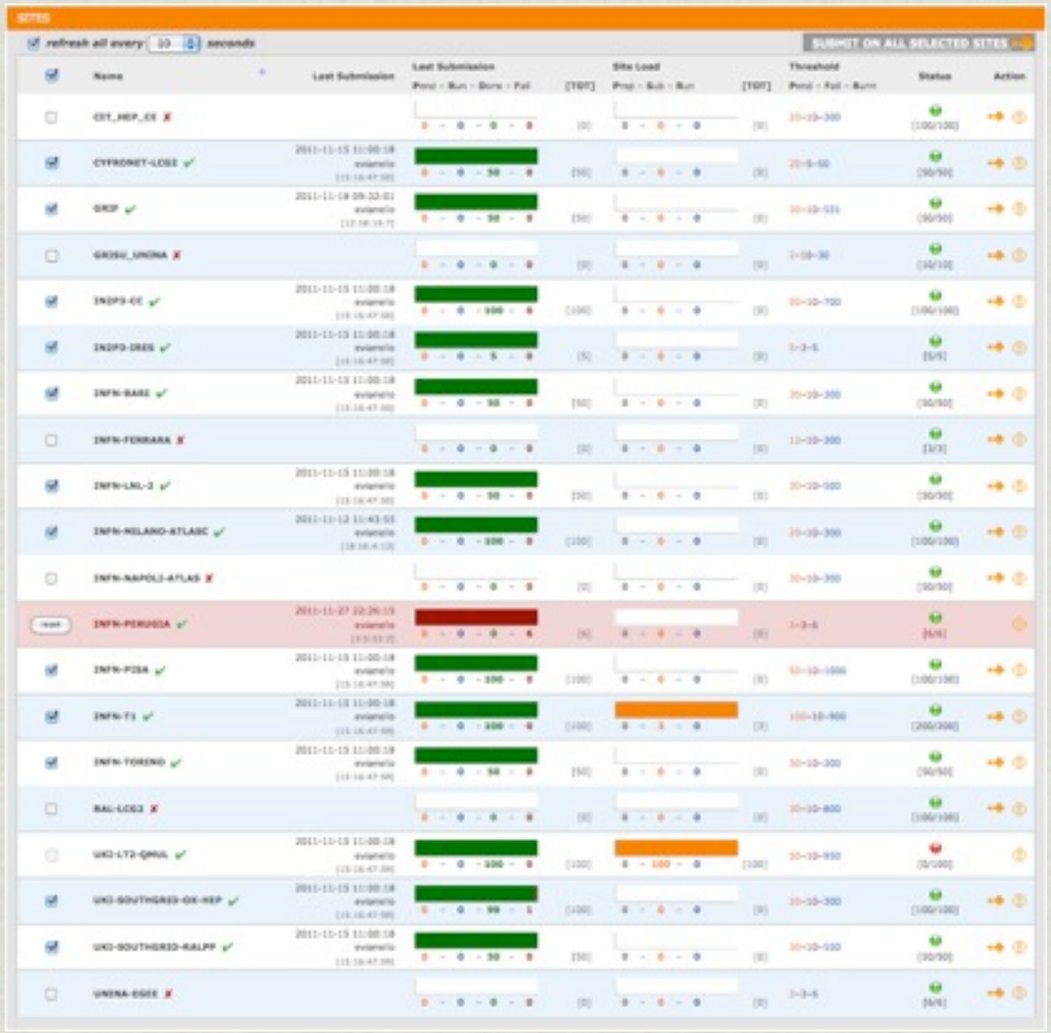
Software Release: V0.2.5 311

Create New Producti

GUIDE

- Type in the new Production Series.
- Specify the release version.
- The system will check for uniqueness of the series. On success, it will create the entry(ies) into the bookkeeping database.

in use...


SITES
refresh all every 10 seconds
SUBMIT ON ALL SELECTED SITES
Name
Last Submission
Last Submission
Site Load
Threshold
Status
Action
CYFRONET-LCG2
GRIF
INFN-BARI
INFN-LNL-2
INFN-MILANO-ATLASC
INFN-PISA
INFN-T1
INFN-TORINO
UKI-LT2-QMUL
UKI-SOUTHGRID-OX-HEP
UKI-SOUTHGRID-RALPP

in use...

FASTSIM CREATE NEW REQUEST

Select custom request and job parameters' values or import them from another production request:

Import From...

production: 2010_July request: [HadRecoilCocktail, B0B0bar_Btag-HD_Cocktail, DG_4, MixSuperbBkg_NoPair] import

Job Parameters:

Analysis: HadRecoilCocktail | Generator: B0B0bar_Btag-HD_Cocktail | Geometry: DG_0 | Background: All Bkg, No Pair (MixSuperb)

A Production Request with the selected parameters doesn't exists and will be created

Request settings:

priority: **10**

minrunnum: 10000000 | required events: 10 | evt/job: 1 | number of jobs: 10

expected event time (sec): 9 | max wct (sec): 57600

Input:

input mode: NONE

Output & Log:

output directory (lfn): lfn:/grid/superbvo.org/production/FastSim/2010_July/output/HadRecoilCocktail/DG_0/B0B0bar_Btag-HD_Cocktail/%RUNNUM%/

output directory (absolute): /production/FastSim/2010_July/output/HadRecoilCocktail/DG_0/B0B0bar_Btag-HD_Cocktail/%RUNNUM%/

log file: /production/FastSim/2010_July/output/HadRecoilCocktail/DG_0/B0B0bar_Btag-HD_Cocktail/log/run%RUNNUM%.log

Edit output directories for the associated test request

test request's output directory (lfn): lfn:/grid/superbvo.org/production/FastSim/2010_July_test/output/HadRecoilCocktail/DG_0/B0B0bar_Btag-HD_Cocktail/%RUNNUM%/

test request's output directory (absolute): /production/FastSim/2010_July_test/output/HadRecoilCocktail/DG_0/B0B0bar_Btag-HD_Cocktail/%RUNNUM%/

test log file: /production/FastSim/2010_July_test/output/HadRecoilCocktail/DG_0/B0B0bar_Btag-HD_Cocktail/log/run%RUNNUM%.log

outputs' sites:

CIT_CMS_T2B	CIT_HEP_CE	CYFRONET-LCG2	GRIF	GRISU_UNINA
IN2P3-CC	IN2P3-IRES	IN2P3-LPSC	INFN-BARI	INFN-CAGLIARI
INFN-FERRARA	INFN-FRASCATI	INFN-LNL-2	INFN-MILANO-ATLASC	INFN-NAPOLI-ATLAS
INFN-PERUGIA	INFN-PISA	☑ INFN-T1	INFN-TORINO	RAL-LCG2
UKI-LT2-QMUL	UKI-SOUTHGRID-OX-HEP	UKI-SOUTHGRID-RALPP	UNINA-EGEE	VICTORIA-LCG2
VICTORIA-LCG2-SL4	WT2			

Other:

note: (max. 255 characters)

CREATE REQUEST

in use...

JOB DETAILS (REQUESTS)

Show submittable requests

Show 10 entries

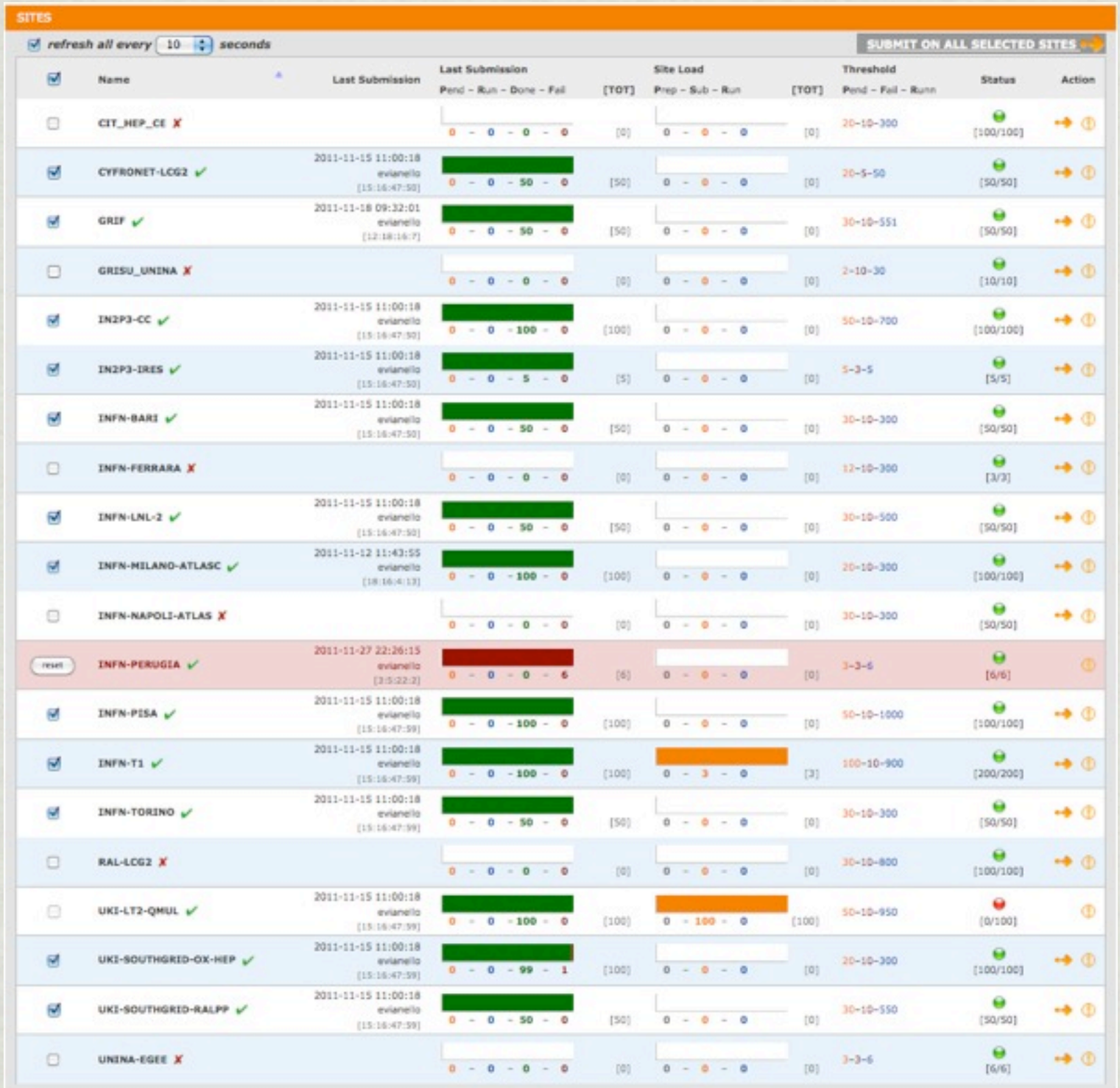
Search:

S	Job Parameters	Priority	Graph Pending – Running – Failed – Done [TOT]	events per job	jobs d/nf/r	events d/nf/r
◉	PRODSCRIPT : ***HadRecoilCocktail*** GENERATOR : ***B0B0bar_SL*** DG : ***DG_4*** TCL : ***PacProduction***	70	0 – 0 – 65 – 2381 [2446]	1	2 381 (6.33%) **2 381** (6.33%) 37 600	2 381 (6.33%) **2 381** (6.33%) 37 600
○	PRODSCRIPT : ***HadRecoilCocktail*** GENERATOR : ***B0B0bar_Btag-HD_Cocktail*** DG : ***DG_4*** TCL : ***MixSuperbBkg_NoPair***	50	0 – 0 – 0 – 5 [5]	80000	5 (0.11%) **5** (0.11%) 4 700	400 000 (0.11%) **400 000** (0.11%) 376 000 000

Showing 1 to 2 of 2 entries

First Previous 1 Next Last

in use...


SITES
refresh all every 10 seconds
SUBMIT ON ALL SELECTED SITES
Name
Last Submission
Last Submission
Pend - Run - Done - Fail
[TOT]
Site Load
Prep - Sub - Run
[TOT]
Threshold
Pend - Fail - Runn
Status
Action
CIT_HEP_CE
CYFRONET-LCG2
GRIF
GRISU_UNINA
IN2P3-CC
IN2P3-IRES
INFN-BARI
INFN-FERRARA
INFN-LNL-2
INFN-MILANO-ATLASC
INFN-NAPOLI-ATLAS
INFN-PERUGIA
INFN-PISA
INFN-T1
INFN-TORINO
RAL-LCG2
UKI-LT2-QMUL
UKI-SOUTHGRID-OX-HEP
UKI-SOUTHGRID-RALPP
UNINA-EGEE

in use...

Automatic submission output

Submission status per site:

```
/data1/script_webui/FastSim/submissionscripts/2010_July/CYFRONET-LCG2_10001643_10001692.php   [32.89 s] log
/data1/script_webui/FastSim/submissionscripts/2010_July/IN2P3-CC_10001693_10001792.php        [52.508 s] log
/data1/script_webui/FastSim/submissionscripts/2010_July/IN2P3-IRES_10001793_10001797.php
/data1/script_webui/FastSim/submissionscripts/2010_July/INFN-BARI_10001798_10001847.php...
/data1/script_webui/FastSim/submissionscripts/2010_July/INFN-LNL-2_10001848_10001897.php...
/data1/script_webui/FastSim/submissionscripts/2010_July/INFN-PISA_10001898_10001997.php...
/data1/script_webui/FastSim/submissionscripts/2010_July/INFN-T1_10001998_10002097.php...
/data1/script_webui/FastSim/submissionscripts/2010_July/INFN-TORINO_10002098_10002147.php...
/data1/script_webui/FastSim/submissionscripts/2010_July/UKI-LT2-QMUL_10002148_10002247.php...
/data1/script_webui/FastSim/submissionscripts/2010_July/UKI-SOUTHGRID-OX-HEP_10002248_10002347.php...
/data1/script_webui/FastSim/submissionscripts/2010_July/UKI-SOUTHGRID-RALPP_10002348_10002397.php...
```

PRODUCTION DATA - SUMMARY

Production Series:	**2010_July**
Software:	**V0.2.5 311**
Test release:	**lfn:/grid/superbvo.org/production/FastSim/2010_September/test_release/V0.2.5_SL-5.3_x86_64_rev311.tjz**

JOB BULK #1 - SUMMARY

Request Parameters:

PRODSCRIPT	GENERATOR	DG	TCL
HadRecoilCocktail	B0B0bar_Btag-HD_Cocktail	DG_4	MixSuperbBkg_NoPair

Jobs' Type	Events per job	Log on db	Target site(s)
normal	80000	no	INFN-T1

#Jobs	Site	First RUNNUM	Last RUNNUM	#Events
50	CYFRONET-LCG2	10002454	10002503	4000000
50	GRIF	10002504	10002553	4000000
100	IN2P3-CC	10002554	10002653	8000000
5	IN2P3-IRES	10002654	10002658	400000
50	INFN-BARI	10002659	10002708	4000000
50	INFN-LNL-2	10002709	10002758	4000000
100	INFN-MILANO-ATLASC	10002759	10002858	8000000
100	INFN-PISA	10002859	10002958	8000000
200	INFN-T1	10002959	10003158	16000000
50	INFN-TORINO	10003159	10003208	4000000
100	UKI-SOUTHGRID-OX-HEP	10003209	10003308	8000000
50	UKI-SOUTHGRID-RALPP	10003309	10003358	4000000
905				**72400000**

Production Request exists! 2010_July

input mode: DIR input path: lfn:/grid/superbvo.org/production/FastSim/2010_September/input

output directory's name: /production/FastSim/2010_July/output/HadRecoilCocktail/DG_4/B0B0bar_Btag-HD_Cocktail/%RUNNUM%

output directory's lfn: lfn:/grid/superbvo.org/production/FastSim/2010_July/output/HadRecoilCocktail/DG_4/B0B0bar_Btag-HD_Cocktail/%RUNNUM%

log file(s): /production/FastSim/2010_July/output/HadRecoilCocktail/DG_4/B0B0bar_Btag-HD_Cocktail/log/run%RUNNUM%.log

Production Initialization and Script Generation ☑ Automatic Submission

in use...

FASTSIM JOB LIST

show filter dialog | refresh last search

Show 10 entries | hide/show columns | Search:

4030 jobs found matching the search criteria.

runnum	events	Analysis	Generator	Geometry	Background	Status	GJid	Site	wct (s)
10002949 evianello	1	HadRecoilCocktail	B0B0bar_Btag-HD_Cocktail	DG_4	MixSuperbBkg_NoPair	done	gjid	CYFRONET-LCG2	289.40
10002948 evianello	1	HadRecoilCocktail	B0B0bar_Btag-HD_Cocktail	DG_4	MixSuperbBkg_NoPair	done	gjid	CYFRONET-LCG2	736.30
10002947 evianello	1	HadRecoilCocktail	B0B0bar_SL	DG_4	PacProduction	done	gjid	CYFRONET-LCG2	997.80
10002946 evianello	1	HadRecoilCocktail	B0B0bar_SL	DG_4	PacProduction	done	gjid	CYFRONET-LCG2	589.90
10002945 evianello	1	HadRecoilCocktail	B0B0bar_SL	DG_4	PacProduction	done	gjid	CYFRONET-LCG2	655.00
10002944 evianello	1	HadRecoilCocktail	B0B0bar_SL	DG_4	PacProduction	done	gjid	CYFRONET-LCG2	737.00
10002943 evianello	1	HadRecoilCocktail	B0B0bar_SL	DG_4	PacProduction	done	gjid	CYFRONET-LCG2	1206.30
10002942 evianello	1	HadRecoilCocktail	B0B0bar_SL	DG_4	PacProduction	done	gjid	CYFRONET-LCG2	897.00
10002941 galvani	1	BtoKNuNu	B+B-_Btag-HD_Cocktail	DG_0	MixSuperbBkg	failed	gjid	INFN-BARI	5.00
10002940 manzali	1	BtoKNuNu	B+B-_Btag-HD_Cocktail	DG_0	MixSuperbBkg	done	gjid	INFN-BARI	89.10

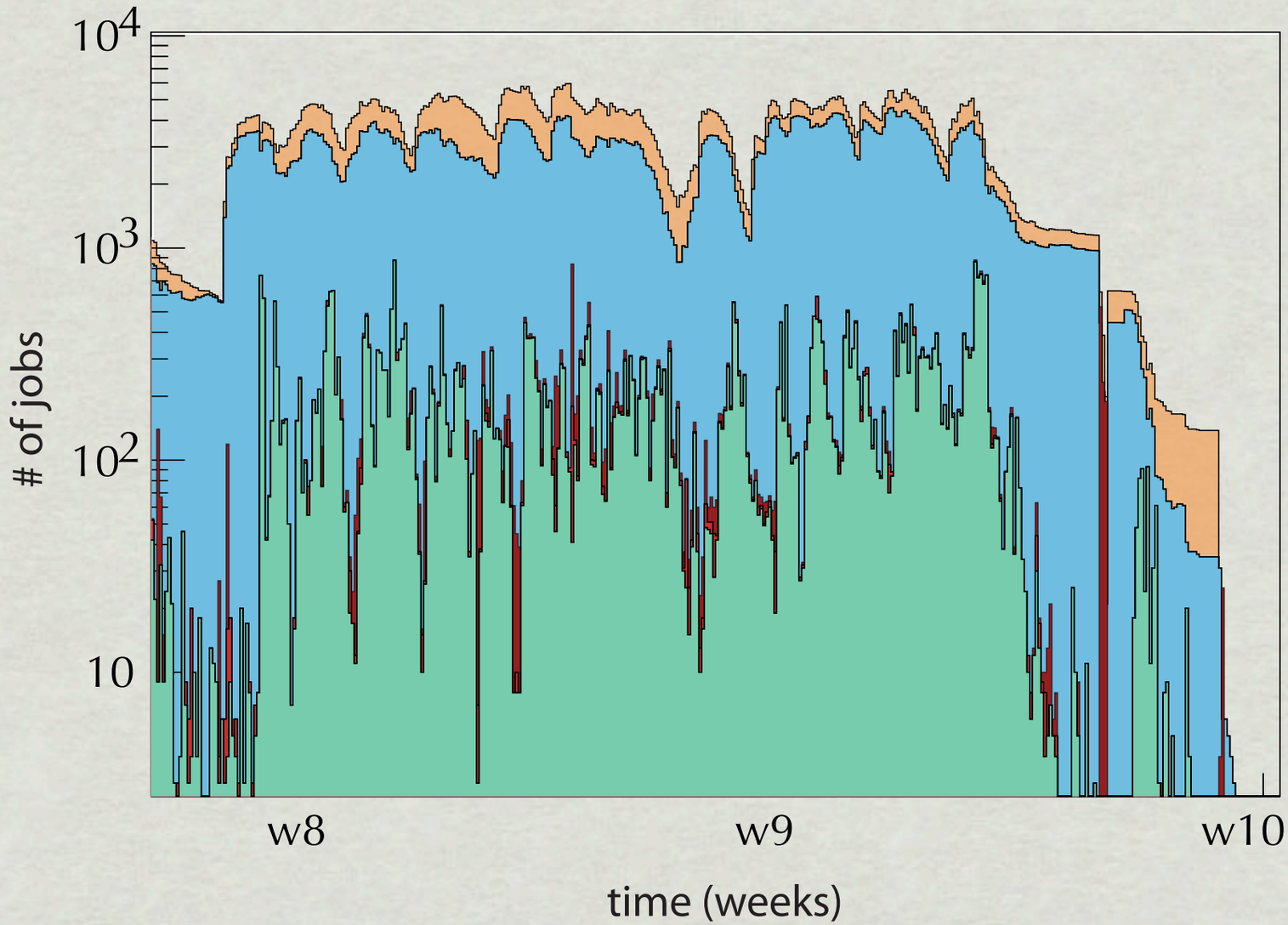
Showing 1,081 to 1,090 of 4,030 entries

First | Previous | 107 | 108 | 109 | 110 | 111 | Next | Last

table generated in '0000:00:02.65' (destroyed in: 0000:00:00.00)

- [x] runnum
- [ ] production
- [ ] uid
- [x] events
- [x] Analysis
- [x] Generator
- [x] Geometry
- [x] Background
- [x] Status
- [ ] Status Reason
- [ ] ErrorMsg
- [x] GJid
- [x] Site
- [ ] ts prepared
- [ ] ts submitted
- [ ] ts running
- [ ] time
- [x] wct (s)

Results



Issues with “production software”

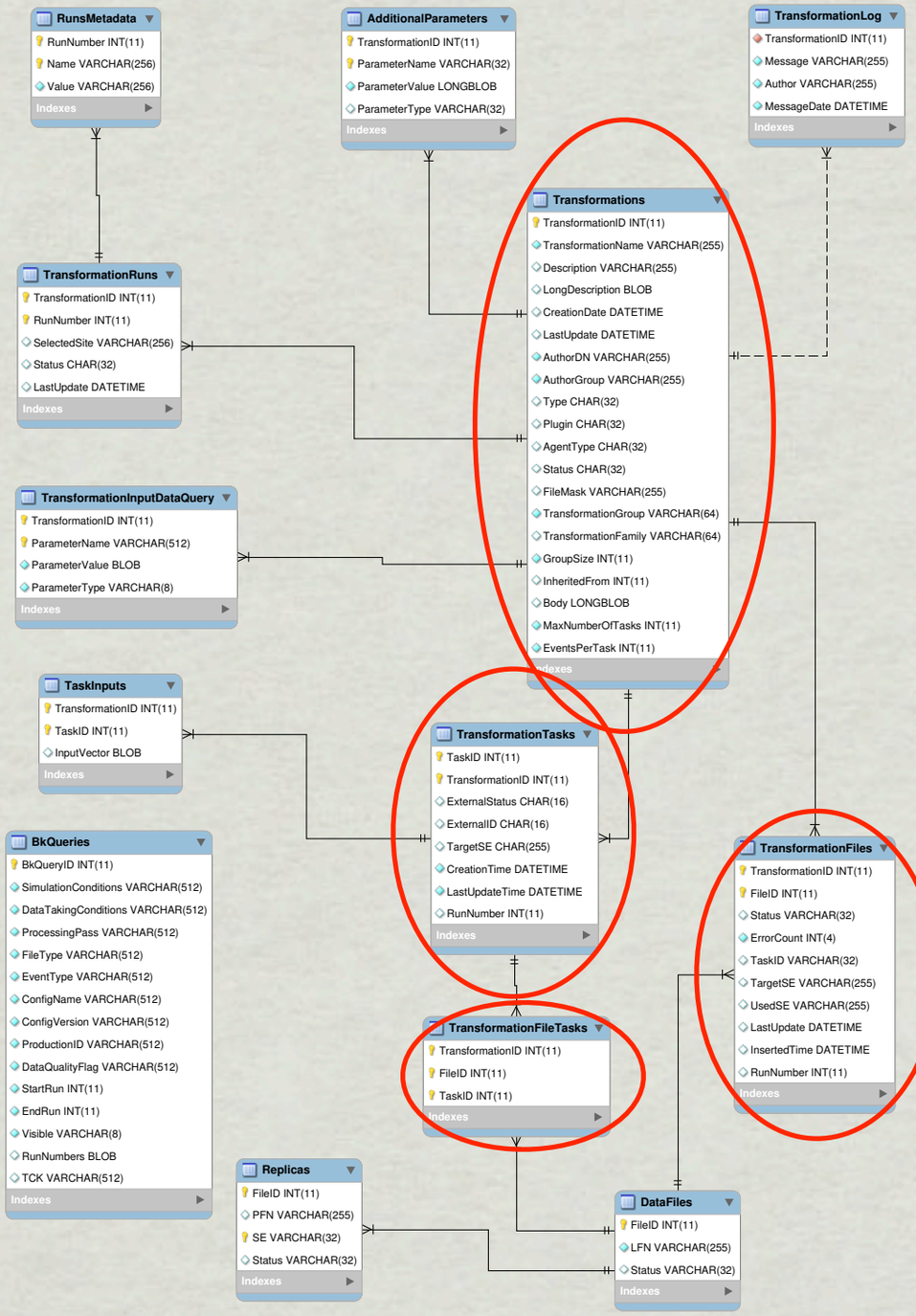
MySQL

- MySQL is one of the most used ‘open-source’ rDBMS
- You may already know it has two main DB engines:
  - **MyISAM**
    These tables have a small footprint. Table-level locking limits the performance in read/write workloads, so it is often used in read-only or read-mostly workloads in Web and data warehousing configurations.
  - **InnoDB**
    A transaction-safe (ACID compliant) storage engine for MySQL that has commit, rollback, and crash-recovery capabilities to protect user data. InnoDB row-level locking (without escalation to coarser granularity locks) and Oracle-style consistent nonlocking reads increase multi-user concurrency and performance. InnoDB stores user data in clustered indexes to reduce I/O for common queries based on primary keys. To maintain data integrity, InnoDB also supports FOREIGN KEY referential-integrity constraints.

MySQL

- To be able to use transactions + take advantage of referential integrity, InnoDB must be used
- If we use DBMS, we should trust them!
  (I understand, sometimes we are paranoic):
  - If we defined a table as
    CREATE TABLE T (ID INT PRIMARY KEY, ...);
    why should we continuously do
    SELECT ID FROM T WHERE ID=<some value>; (to check for existence)
    before doing
    UPDATE T SET ... WHERE ID=<some value>;?
    or
    INSERT INTO T2 (..., IDfk, ...) VALUES (..., <some value>,...);
    where IDfk is defined as Foreign Key referencing ID in T?

DIRAC Transformation System



SHOW TABLE STATUS

```
           Name: Transformations
         Engine: InnoDB
        Version: 10
     Row_format: Compact
           Rows: 102398
 Avg_row_length: 6303
    Data_length: 645447680
Max_data_length: 0
   Index_length: 4767744
      Data_free: 4194304
 Auto_increment: 24618
    Create_time: 2010-05-18 13:24:05
    Update_time: NULL
     Check_time: NULL
      Collation: latin1_swedish_ci
       Checksum: NULL
 Create_options:
        Comment:
```

SHOW TABLE STATUS

```
           Name: TransformationTasks
         Engine: MyISAM
        Version: 10
     Row_format: Fixed
           Rows: 3244338
 Avg_row_length: 316
    Data_length: 1218232456
Max_data_length: 8894609264056729 5
   Index_length: 109328384
      Data_free: 193021648
 Auto_increment: 1
    Create_time: 2010-05-27 12:23:07
    Update_time: 2013-06-04 18:34:44
     Check_time: 2010-05-27 12:23:09
      Collation: latin1_swedish_ci
       Checksum: NULL
 Create_options:
        Comment:
```

SHOW TABLE STATUS

```
           Name: TransformationFiles
         Engine: MyISAM
        Version: 10
     Row_format: Dynamic
           Rows: 16451371
 Avg_row_length: 91
    Data_length: 1511429528
Max_data_length: 281474976710655
   Index_length: 549894144
      Data_free: 0
 Auto_increment: NULL
    Create_time: 2010-05-27 12:23:03
    Update_time: 2013-06-04 18:35:33
     Check_time: 2010-05-27 12:23:07
      Collation: latin1_swedish_ci
       Checksum: NULL
 Create_options:
        Comment:
```

SHOW TABLE STATUS

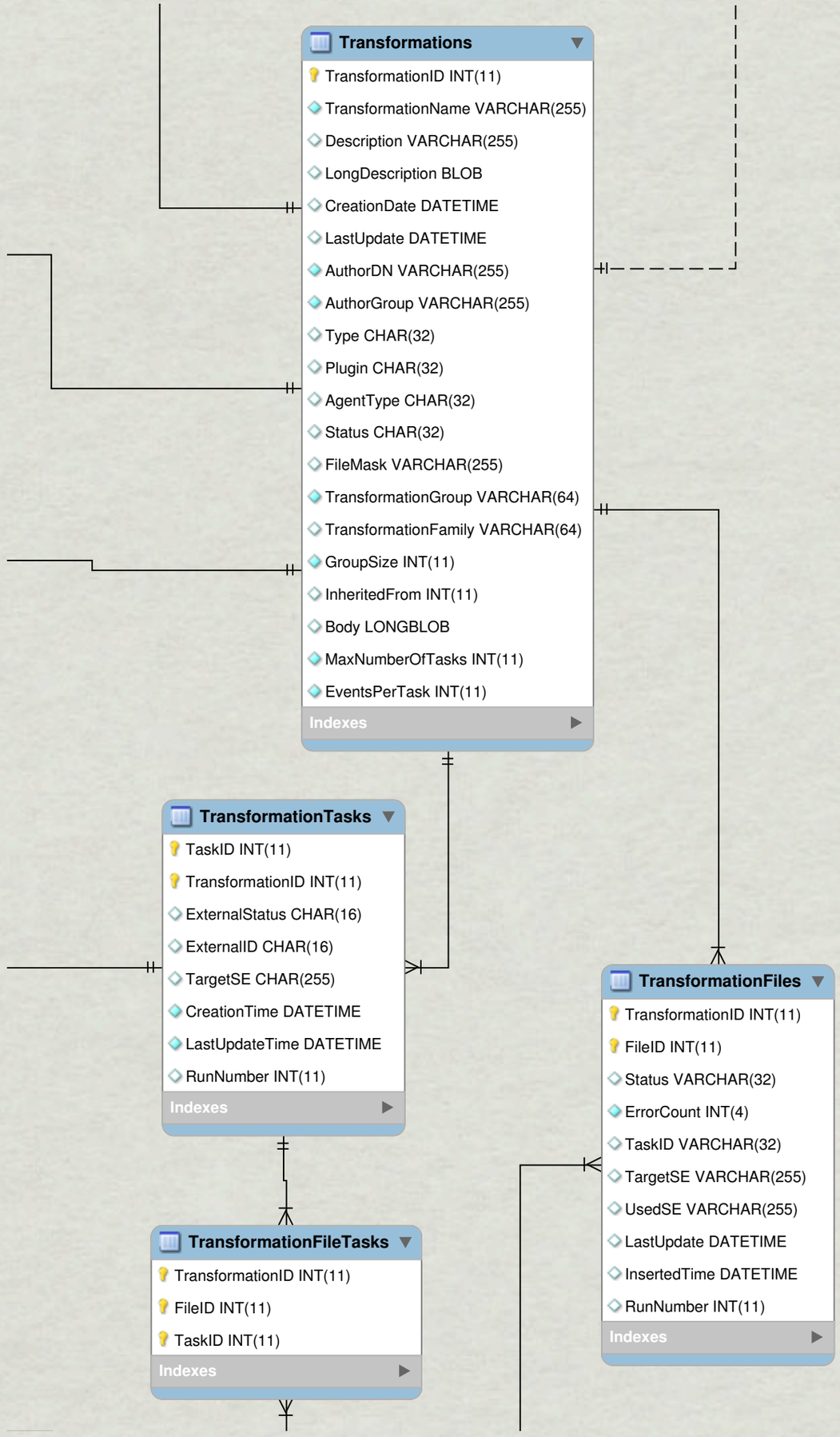
```
           Name: TransformationFileTasks
         Engine: MyISAM
        Version: 10
     Row_format: Fixed
           Rows: 63564637
 Avg_row_length: 13
    Data_length: 826340281
Max_data_length: 3659174697238527
   Index_length: 1405467648
      Data_free: 0
 Auto_increment: NULL
    Create_time: 2010-08-09 20:53:22
    Update_time: 2013-06-04 18:35:50
     Check_time: NULL
      Collation: latin1_swedish_ci
       Checksum: NULL
 Create_options:
        Comment:
```

SHOW TABLE STATUS

```
           Name: DataFiles
         Engine: MyISAM
        Version: 10
     Row_format: Dynamic
           Rows: 50722343
 Avg_row_length: 90
    Data_length: 4581560000
Max_data_length: 281474976710655
   Index_length: 7016133632
      Data_free: 0
 Auto_increment: 50722344
    Create_time: 2010-02-15 22:55:18
    Update_time: 2013-06-04 18:35:10
     Check_time: NULL
      Collation: latin1_swedish_ci
       Checksum: NULL
 Create_options:
        Comment:
```

Clean-up

- Assuming all table InnoDB
- All FK defined
- To delete finished tasks (+files+...)
  - is there some order in the DELETE operations?
  - may we use the CASCADE OPTION?



Slow queries

- Some query may require long time to execute...
  - needs for indexes
  - modify db design or application logic?
- Log analysis is crucial to figure out the problems

Slow queries

```
# Files: 22_slow.log
# Overall: 2.57M total, 176 unique, 206.73 QPS, 0.98x concurrency ________
# Time range: 2013-04-16 18:31:40 to 21:59:14
# Attribute          total     min     max     avg     95%  stddev  median
# ============     ======= ======= ======= ======= ======= ======= =======
# Exec time          12224s     1us    114s     5ms   316us   386ms    16us
# Lock time           6342s       0     96s     2ms    44us   345ms        0
# Rows sent          11.49M       0  12.46k    4.68    0.99   94.10        0
# Rows examine      754.03M       0  10.82M  307.09    0.99  20.90k        0
# Query size        159.38M       6  79.15k   64.91  212.52  385.80    26.08

# Profile
# Rank Query ID           Response time   Calls   R/Call  V/M   Item
# ==== ================== =============== ======= ======= ===== ==========
#    1 0xB6181EDD2FB888A8 6400.6257 52.4%   11330  0.5649 27.07 SELECT TransformationFiles
#    2 0x874051C92A574963  878.2083  7.2%   30592  0.0287 40.62 UPDATE TransformationFiles
#    3 0xAFA91D2D2DCA4EA4  804.9874  6.6%       9 89.4430  8.12 SELECT TransformationFiles
#    4 0xACF49C16D92EB3A8  642.1562  5.3%    9741  0.0659 27.56 SELECT TransformationFiles
#    5 0xA1FB762B8E26019C  479.3516  3.9%    3632  0.1320 42.20 SELECT TransformationFiles
#    6 0x32101DCF176F3EDA  423.3719  3.5%   29471  0.0144  0.06 SELECT TransformationTasks
#    7 0x12A6600A7E70CB2E  312.4006  2.6%    1411  0.2214  0.00 SELECT Transformations
#    8 0x2B26517093495FE9  292.0841  2.4%    1304  0.2240  0.01 SELECT Transformations
#    9 0x7E3A04D71EF9BF1B  255.9953  2.1%     217  1.1797 75.80 SELECT TransformationFiles
...
```

Slow queries

```
Filter with WHERE
+- Bookmark lookup
   +- Table
   |  table          TransformationFiles
   |  possible_keys  Status
   +- Index range scan
      key            TransformationFiles->Status
      possible_keys  Status
      key_len        35
      rows           11334650
```

```
# Query 3: 0.03 QPS, 2.55x concurrency, ID 0xAFA91D2D2DCA4EA4 at byte 313302282
# This item is included in the report because it matches --limit.
# Scores: V/M = 8.12
# Time range: 2013-04-16 20:26:12 to 20:31:28
# Attribute    pct   total     min     max     avg     95%  stddev  median
# ============ === ======= ======= ======= ======= ======= ======= =======
# Count          0       9
# Exec time      6    805s     51s    114s     89s    113s     27s    102s
# Lock time      2    157s    44us     53s     17s     52s     24s   134ms
# Rows sent      0      56       1      18    6.22   17.65    6.24    3.89
# Rows examine  12  97.38M  10.82M  10.82M  10.82M  10.76M       0  10.76M
# Query size     0   2.53k     225     429  287.67  420.77   72.04  258.32
# String:
# Databases    ProductionDB
# Hosts        volhcb19.c... (4/44%), volhcb29.c... (4/44%), volhcb24.c... (1/11%)
# Users        Dirac
# Query_time distribution
#  10ms
# 100ms
#    1s
#  10s+  ################################################################
# Tables
#    SHOW TABLE STATUS FROM `ProductionDB` LIKE 'TransformationFiles'\G
#    SHOW CREATE TABLE `ProductionDB`.`TransformationFiles`\G
# EXPLAIN /*!50100 PARTITIONS*/
SELECT
TransformationID,FileID,Status,TaskID,TargetSE,UsedSE,ErrorCount,LastUpdate,InsertedTime,RunNumber
FROM TransformationFiles    WHERE `Status` IN ( "Processed", "Problematic" ) AND `FileID` IN
( "29643136", "29643865", "35351500", "29826552", "35351597" ) LIMIT 10000\G
```

Conclusions

- Software for Physics is generally written by physicists (also Experiment's Core Software)
- so... if you need to interface your software to or base your software on databases: you must 'learn' databases, especially database design
- Interaction with experts is recommended for both db design and programming techniques

Credits

- Elmasri and Navathe
  Fundamentals of Database Systems, 5/6th Ed.
  Pearson, Addison Wesley
- E. Vianello, A. Fella, E. Luppi
- DIRAC and LHCbDIRAC developers

Notice 2

- I left out:
  - Relational Calculus
  - Functional Dependencies and **Normalization**
  - Transactions, Concurrency, Recovery, Physical level, ...
  - all the emerging non-relational databases (NoSQL), such as MongoDB, CouchDB, Hadoop-based, ...