Chapter 1

* Database: organized collection of logically related data
* Data: stored representations of meaningful objects and events 🡪 structured and un---
* Information: data processed to increase knowledge in the person using the data
* Metadata: data that describes the properties and context of user dat
* Disadvantages of file processing 🡪 this is the advantages of data base
* Program-Data Dependence 🡪 each program
* maintain his/her own data
* needs to include code for the metadata
* Non-standard file formats
* Duplication of Data
* Waste of space
* Data changes in one file could cause inconsistencies
* Limited Data Sharing
* Lengthy Development Times
* Excessive Program Maintenance
* Database cost and risk
* New, specialized personnel
* Installation and management cost
* Conversion costs
* Need for explicit backup and recovery
* Database Management System: A software system that is used to create, maintain, and provide controlled access to user databases
* Elements of database: data models – relational databases – use of internet – database applications 🡪 programs do activities create – delete – update database
* Data Warehouse: Integrated decision support system derived from various operational databases
* Enterprise Resource Planning (ERP): Integrate all enterprise functions
* Range of data base applications: personal – workgroup – departmental - enterprice
* Components of data base environment:
* CASE Tools–computer-aided software engineering
* Repository–centralized storehouse of metadata
* Database Management System (DBMS) / Database
* Application Program / User Interface
* Data/Database Administrators–personnel responsible for maintaining the database
* System Developers–personnel responsible for designing databases and software
* End Users

Chapter 2

* Enterprise Data Model: first step – specify scope – specify entity types and relations
* Information Systems Architecture ISA: Conceptual blueprint for organization’s desired information systems structure
* Data - ERD
* Processes – DFD
* Data Network–topology diagram
* Events
* Reasons for events and rules
* CASE: Computer-Aided Software Engineering –software tools providing automated support for systems development
* Data modeling / Code generation / Repositories
* Packaged Data Models: Model components that can be customized, and assembled into full-scale data models 🡪 Universal / Industry-specific data models
* Reduced development time
* Higher model quality
* Information Engineering: A data-oriented methodology to create and maintain information systems
* Three-tiered client/web App/ database enterprise server database architecture
* Top-down planning–a generic IS planning methodology for obtaining a broad understanding of the IS needed by the entire organization
* Planning 🡪 align information technology with organization’s business strategies
* enterprise modeling and early conceptual data modeling

1. Identify strategic planning factors

* Organization goals
* Critical success factors
* Problem areas –weaknesses

1. Identify corporate planning objects

* Organizational units
* Organizational locations
* Business functions
* Entity types
* Information systems

1. Develop enterprise model

* Functional decomposition 🡪 break system description into finer
* Enterprise data model
* Planning matrixes 🡪 interrelationships between planning objects

1. Function-to-data entity
2. Location-to-function
3. Unit-to-function
4. IS-to-data entity

* Analysis 🡪 integrated conceptual data modeling / requirements analysis
* Logical Design 🡪 transactions, forms, information requirement structure, data integrity and security
* Physical design 🡪 develop technology and organizational specifications like DBMS
* Implementation 🡪 including coded programs, documentation, installation, training and conversion
* Maintenance 🡪 monitor, repair, enhance and periodic audits

Chapter 7 - select

* Scalar aggregate: single value returned from SQL query with aggregate function
* Vector aggregate: multiple values returned from SQL query ith aggregate function (via GROUP BY)
* Database Schema: 🡪 start with abstract views then add details
* External Schema 🡪 User Views / subset of conceptual schema / repeats entity
* specific representation of an external view
* Easy to identify specific data required
* Facilitates designer’s job
* Ensures security
* Simplifies application development
* Conceptual Schema 🡪 E-R models
* Represents global view of the entire database
* All external views integrated into single global view
* Provides a relatively easily understood macro level view
* Independent of both software DBMS and hardware
* Internal model 🡪 Representation of the database as “seen” by the DBMS / SQL
* Maps the conceptual model to the DBMS / depends on software
* Logical independence: change internal model without affecting conceptual model
* Physical Schema 🡪 Physical structures
* Operates at lowest level of abstraction / the way data are saved on storage
* Physical independence: changes in physical model do not affect internal model

Chapter 3

* Business Rules: Statements that define some aspect of the business / Control business behavior / Automated through DBMS software
* Declarative/what not how - Distinct - Business-oriented – Expressible – Precise/clear
* Good data name: Related to business / Meaningful / Unique / Readable
* Term – word or phrase with specific meaning
* Fact – association between two or more terms
* Relationship degrees specify number of entity types involved 🡪 unary / binary / ternary
* Relationship cardinalities specify how many of each entity type is allowed
* mandatory / optional 🡪 one / many
* Cardinality Constraints - the number of instances of one entity that can or must be associated with each instance of another entity
* Entity should be 🡪 important in system - multi value - multi attribute
* Entity should not be 🡪 A user / An output of the database system
* Attribute – property or characteristic of an entity or relationship type
* Required versus Optional Attributes
* Simple versus Composite Attribute
* Single-Valued versus Multivalued Attribute
* Stored versus Derived Attributes
* Identifier Attributes
* Associative Entity – combination of relationship and entity - a relationship with an attribute
* Ternary relationships and many-to-many relationships should be converted to associative entity
* Identifier (Key)–An attribute that uniquely identifies individual instances
* Will not change in value - Will not be null - No intelligent
* Candidate Identifier – an attribute that could be a key … satisfies the requirements for being an identifier
* Alternate Key: For example, in an employee table, empno is a primary key, empname is a alternate key that may not be unique but still helps in identifying a row of the table.
* Strong entities
* exist independently of other types of entities
* has its own unique identifier
* identifier underlined with single-line
* Weak entity
* dependent on a strong entity
* does not have a unique identifier
* Partial identifier underlined with double-line
* Entity box has double line
* Identifying relationship: links strong entities to weak entities

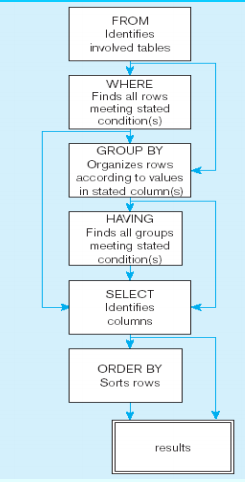
Chapter 4

* Sub-type: A subgrouping of the entities in an entity type that has attributes distinct from those in other subgroupings
* Super-type: A generic entity type that has a relationship with one or more subtypes
* An instance of a subtype is also an instance of the super-type
* Relationships at the super-type level indicate that all subtypes will participate in the relationship
* Generalization: The process of defining a more general entity type from a set of more specialized entity types. BOTTOM-UP
* Specialization: The process of defining one or more subtypes of the super-type and forming super-type/subtype relationships. TOP-DOWN
* Completeness Constraints: Whether an instance of a super-type must also be a member of at least one subtype
* Total Specialization Rule 🡪 double line 🡪 yes
* Partial Specialization Rule 🡪 single line 🡪 no
* Disjointness Constraints: Whether an instance of a super-type may simultaneously be a member of two (or more) subtypes
* Disjoint Rule 🡪 from only one subtype 🡪 d inside
* Overlap Rule 🡪 from more than one subtype 🡪 o inside
* Subtype Discriminator: An attribute of the super-type whose values determine the target subtype(s)
* Disjoint – a simple attribute 🡪 selects one sub-type
* Overlapping – a composite attribute 🡪 selects group of sub-types
* Entity cluster: Set of one or more entity types and associated relationships grouped into a single abstract entity type
* Classification of business rules:
* Derivation–rule derived from other knowledge
* Structural assertion–rule expressing static structure
* Action assertion–rule expressing controls of organizational actions
* Anchor Object – an object on which actions are limited
* Corresponding Objects – an object influencing the ability to perform an action on another business rule
* Types of action assertion
* Result from assertion 🡪 condition / integrity / authorization
* Form from assertion 🡪 enabler / timer / executive
* Rigor of the assertion 🡪 controlling / influencing

Chapter 7

* RDBMS: A database management system that manages data as a collection of tables in which all relationships are represented by common values in related tables
* Purpose of SQL 🡪 define data structures – standards and its enhancement – portability
* Benefits of standardization 🡪 reduce training cost – portability – productivity –longevity
* Catalog: A set of schemas that constitute the description of a database
* Schema: The structure that contains descriptions of objects created by a user
* Data Definition Language (DDL): Commands that define a database
* Data Manipulation Language (DML) Commands that maintain and query a database
* Data Control Language (DCL) Commands that control a database, including administering privileges and committing data
* Views provide users controlled access to tables
* Dynamic View🡪A “virtual table” created dynamically upon request - No data actually stored
* Materialized View🡪Copy or replication of data - Data actually stored – refreshed periodically
* Simplify query commands
* Enhance programming productivity
* Use little storage space
* May or may not be directly updateable – processing time is reference
* DBMS uses Indexes to Speed up random/sequential access to base table data

Chapter 8

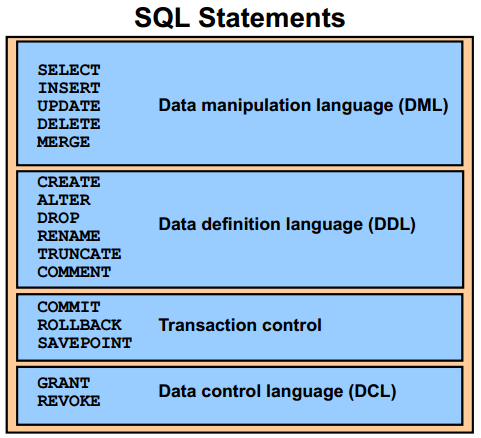
* Sub query – placing an inner query (SELECT statement) inside an outer query
* Non-correlated – executed once for the entire outer query – doesn’t depend on outer
* Correlated – executed once for each row returned by the outer query – uses exits
* Best make sub-query in from statement
* Transaction = A discrete unit of work that must completely processed or not processed at all
* Functions – routines that return values and take input parameters
* Procedures – routines that do not return values and can take input or output parameters
* Triggers - Routines that execute in response to a database event

Chapter 5

* A relation is a named, two-dimensional table of data in 1st normal form
* For a table to be a relation
* Has unique name
* Each column unique name
* Each row is unique
* The order of columns and row is irrelevant
* Foreign keys: identifiers that enable a dependent relation to refer to its parent relation
* alternate key: when two attributes can be used as primary key and we can use any of them one of them is primary key and the other one is the alternate key the two are called candidate ID
* integrity constraints:
* domain constraint 🡪 set of values assigned to attribute 🡪 name , type , size
* entity integrity 🡪 must have primary key , not to be null
* referential integrity 🡪 any foreign key value must match a primary key value
* delete rules
* restrict 🡪 don’t delete parent side of dependent side
* cascade 🡪 delete dependent side with the parent side
* set-to-null 🡪 set foreign key to null when delete parent side

Data Normalization

* Primarily a tool to validate and improve a logical design so that it satisfies certain constraints that avoid unnecessary duplication of data
* A relation that contains minimal data redundancy and allows insertion , deletion without causing data inconsistencies
* Goal is to avoid anomalies:
* Insertion Anomaly –adding new rows forces user to create duplicate data
* Deletion Anomaly –deleting rows may cause a loss of data that would be needed for other future rows
* Modification Anomaly –changing data in a row forces changes to other rows because of duplication
* A table should not pertain to more than one entity type
* Functional Dependency: The value of one attribute (the determinant) determines the value of another attribute
* Why do these anomalies exist?
* Because there are multiple themes (entity types) in one relation. This results in duplication and an unnecessary dependency between the entities
* every non-key attribute is fully functionally dependent on the ENTIRE primary key no partial functional dependency
* transitive dependencies (functional dependencies on non-primary-key attributes)



Data manipulation language

# Select

* **arithematic opeartion**

**SELECT last\_name, salary, salary + 300**

**FROM employees;**

* **alias**

**SELECT last\_name "Name" , salary\*12 "Annual Salary"**

**FROM employees;**

* **concatenation operator**

**SELECT last\_name || job\_id AS "Employees"**

**FROM employees;**

* **alternatve quote**

**SELECT department\_name ||**

**q'[, it's assigned Manager Id: ]'**

**|| manager\_id**

**AS "Department and Manager"**

**FROM departments;**

**// to get any created table data**

**desc table\_name**

**// to get all tables owned by user connect it then**

**select tablespace\_name, table\_name from user\_tables;**

**Select \* from tab ;**

* **where with advanced work**

**SELECT last\_name, salary**

**FROM employees**

**WHERE salary <> 3000**

**AND salary BETWEEN 2500 AND 3500**

**AND manager\_id IN (100, 101, 201)**

**ANd first\_name LIKE 'S%'**

**OR manager\_id IS NULL**

**or job\_id**

**NOT IN ('IT\_PROG', 'ST\_CLERK', 'SA\_REP') ;**

* **using scape operator to search for %**

**select mycol from mytable where mycol like '%\%%';**

**// here we scaped the middle %**

* **order by**

**SELECT last\_name, department\_id, salary**

**FROM employees**

**ORDER BY department\_id, salary DESC;**

* **subustitution variable**

**SELECT employee\_id, last\_name, salary, department\_id &column\_name**

**FROM employees**

**WHERE &condition**

**or employee\_id = &employee\_num**

**or job\_id = '&job\_title'**

**order by &order\_column;**

* **reuse the substitiution variable**

**SELECT employee\_id, last\_name, job\_id, &&column\_name**

**FROM employees**

**ORDER BY &column\_name ;**

* **select with column number and order with one up and one down**

**SELECT LNAME, JOB, SALARY FROM trainer**

**WHERE STATE = ‘Giza’**

**ORDER BY 3 DESC, 2;**

* **select ‘fixed\_string’ from tabel\_name;**
* **correlated sub-query with exists 🡪 returns false for empty set**

**SELECT DISTINCT ORDER\_ID FROM ORDER\_LINE\_T**

**WHERE EXISTS (SELECT \* FROM PRODUCT\_T**

**WHERE PRODUCT\_ID = ORDER\_LINE\_T.PRODUCT\_ID**

**AND PRODUCT\_FINISH = ‘Natural ash’);**

* **SELECT PRODUCT\_DESCRIPTION, STANDARD\_PRICE, AVGPRICE**

**FROM (SELECT AVG(STANDARD\_PRICE) AVGPRICE FROM PRODUCT\_T),**

**PRODUCT\_T**

**WHERE STANDARD\_PRICE > AVGPRICE;**

* **sub-query in the from is prefered over this one in where**
* **define variabel**

**DEFINE employee\_num = 200**

* **woring with date**

**SELECT last\_name, hire\_date FROM employees**

**WHERE hire\_date < '01-FEB-88';**

* **Subtract two dates to find the number of days between them**
* **distinct with select statements so you can select unique coloum values or unique total row values**

**//DUAL is an empty table that we can use to do operation on select;**

* **SQL functions:**
* **single functions**
* **character**
* **LOWER**
* **UPPER**
* **INITCAP**
* **CONCAT('Hello', 'World')**
* **SUBSTR('HelloWorld',1,5)**
* **LENGTH('HelloWorld')**
* **INSTR('HelloWorld', 'W')**
* **LPAD(salary,10,'\*')**
* **RPAD(salary, 10, '\*')**
* **REPLACE ('JACK and JUE','J','BL')**
* **TRIM('H' FROM 'HelloWorld'**
* **numbers**
* **ROUND(45.926, 2)**
* **TRUNC(45.926, 2)**
* **MOD(1600, 300)**
* **date**
* **MONTHS\_BETWEEN ('01-SEP-95','11-JAN-94')**
* **ADD\_MONTHS ('11-JAN-94',6)**
* **NEXT\_DAY ('01-SEP-95','FRIDAY')**
* **LAST\_DAY ('01-FEB-95')**
* **ROUND(SYSDATE,'MONTH')**
* **ROUND(SYSDATE ,'YEAR')**
* **TRUNC(SYSDATE ,'MONTH')**
* **TRUNC(SYSDATE ,'YEAR')**
* **TO\_CHAR(number, 'format\_model')**
* **TO\_CHAR(DATE, 'format\_model')**
* **TO\_NUMBER(char[, 'format\_model'])**
* **TO\_DATE(char[, 'format\_model'])**
* **general**
* **NVL (expr1, expr2)**
* **NVL2 (expr1, expr2, expr3)**
* **NULLIF (expr1, expr2)**
* **COALESCE (expr1, expr2, ..., exprn)**
* **CASE job\_id WHEN 'IT\_PROG' THEN 1.10\*salary**

**WHEN 'ST\_CLERK' THEN 1.15\*salary**

**WHEN 'SA\_REP' THEN 1.20\*salary**

**ELSE salary END**

* **DECODE(job\_id, 'IT\_PROG', 1.10\*salary,**

**'ST\_CLERK', 1.15\*salary,**

**'SA\_REP', 1.20\*salary,**

**salary)**

* **group function**
* **AVG**
* **COUNT 🡪 counts number of non null rows / can use distinct**
* **MAX 🡪 can be used with characger , number and date**
* **MIN 🡪 can be used with characger , number and date**
* **SUM**
* **constraint any attribute writen beside the group function should be added in the group by**

**select department\_id , avg (salary ) from employees**

**group by department\_id ;**

* **group with two paramaters**

**SELECT department\_id dept\_id, job\_id, SUM(salary)**

**FROM employees**

**GROUP BY department\_id, job\_id**

**HAVING MAX(salary)>10000 ;**

* **table in anther user**

**SELECT \* FROM userA.employees;**

# Update

* **UPDATE employees**

**SET department\_id = 55**

**WHERE department\_id = 110;**

**Delete 🡪 if we don’t use where clause it will delete all rows**

* **DELETE FROM departments**

**WHERE department\_id = 60;**

* **delete with sub-quary**
* **DELETE FROM employees**

**WHERE department\_id =**

**(SELECT department\_iD FROM departments**

**WHERE department\_name LIKE '%Public%');**

* **we should commit after we update , delete , insert in the data so we keep changes we made in data**

# Insert

* **INSERT INTO departments(department\_id,**

**department\_name, manager\_id, location\_id)**

**VALUES (70, 'Public Relations', 100, 1700);**

**// Optionally, list the columns in the INSERT clause**

* **insert with sub-quary 🡪 we don’t use values clause**
* **INSERT INTO sales\_reps(id, name, salary, commission\_pct)**

**SELECT employee\_id, last\_name, salary, commission\_pct**

**FROM employees**

**WHERE job\_id LIKE '%REP%';**

Data Definition language

**/ if two entities has dependent relations**

**//we need to make them**

**//then we alter one of them with the new relation**

**// we create tables in order of dependency**

**// then we check the constrains of each data**

**create table job(**

**jo\_id varchar2(3) primary key ,**

**job\_No varchar2(20) default 1200 ,**

**minSal number(4) unique not null ) ;**

* **creating a table using sub-query**

**CREATE TABLE dept80**

**AS**

**SELECT employee\_id, last\_name,**

**salary\*12 ANNSAL,**

**hire\_date**

**FROM employees**

**WHERE department\_id = 80;**

* **constrains in table creation**
* **primary key**
* **foreign key**
* **not null**
* **unique**
* **checks**
* **creatig index**

**Create index Name\_IDX on Customer\_T(Customer\_Name);**

* **deleting index**

**Drop Index Name\_IDX;**

* **name constraint in the same line**
* **employee\_id NUMBER(6) CONSTRAINT emp\_emp\_id\_pk PRIMARY KEY**
* **Name constraint in the last line**
* **CONSTRAINT emp\_emp\_id\_pk PRIMARY KEY (EMPLOYEE\_ID)**
* **CONSTRAINT emp\_email\_uk UNIQUE(email)**
* **constraint emp\_pri primary key (emp\_id , emp\_name)**
* **check ( emp\_name in ( ‘Mahmoud ’ , ‘ALi’) )**
* **foreign key naming in the last line**
* **CONSTRAINT emp\_dept\_fk FOREIGN KEY (department\_id) REFERENCES departments(department\_id)**
* **check input range for an attribute**

**CONSTRAINT emp\_salary\_min CHECK (salary > 0)**

**//to delete a table**

**//You cannot roll back the DROP TABLE statement**

**DROP TABLE dept80;**

**Alter**

* **ALTER TABLE department DROP COLUMN head\_id ;**
* **ALTER TABLE department ADD head\_id number(11) REFERENCES employee(emp\_id);**
* **ALTER TABLE table\_name RENAME TO new\_table\_name;**
* **ALTER TABLE supplier MODIFY supplier\_name varchar2(100) not null;**
* **ALTER TABLE supplier RENAME COLUMN supplier\_name to sname;**

# Truncate 🡪 delete all rows in tables and leave it empyt

* **TRUNCATE TABLE copy\_emp;**

All join types

**SELECT table1.column, table2.column**

**FROM table1**

**[NATURAL JOIN table2] |**

**[JOIN table2 USING (column\_name)] |**

**[JOIN table2 ON (table1.column\_name = table2.column\_name)]|**

**[LEFT|RIGHT|FULL OUTER JOIN table2 ON (table1.column\_name = table2.column\_name)]|**

**[CROSS JOIN table2];**

* **using table alaisees**

**SELECT e.employee\_id, e.last\_name, d.location\_id, department\_id**

**FROM employees e JOIN departments d**

**USING (department\_id);**

* **join with additional conditions**

**SELECT e.employee\_id, e.last\_name, e.department\_id, d.department\_id, d.location\_id**

**FROM employees e JOIN departments d**

**ON (e.department\_id = d.department\_id)**

**AND e.manager\_id = 149;**

* **multiple join**

**SELECT employee\_id, city, department\_name**

**FROM employees e**

**JOIN departments d**

**ON d.department\_id = e.department\_id**

**JOIN locations j**

**ON e.salary BETWEEN j.lowest\_sal AND j.highest\_sal;**

* **outer join get data of matched and unmatche data**

**SELECT e.last\_name, d.department\_id, d.department\_name**

**FROM employees e FULL/left/right OUTER JOIN departments**

**ON (e.department\_id = d.department\_id);**

Sub-queries

* **SELECT last\_name FROM employees**

**WHERE salary >**

**(SELECT salary FROM employees WHERE last\_name = 'Abel');**

* **sub-querie with havingg**

**SELECT department\_id, MIN(salary)**

**FROM employees GROUP BY department\_id**

**HAVING MIN(salary) >**

**(SELECT MIN(salary)FROM employees WHERE department\_id = 50);**

* **multiple sub-query using ANY / ALL / in**

**SELECT employee\_id, last\_name, job\_id, salary FROM employees**

**WHERE salary < ALL/ANY**

**(SELECT salary FROM employees WHERE job\_id = 'IT\_PROG');**

Data control language

**clear screen**

**// any command must end with semi-colon**

**// user can add new database**

**// but admin has rule to adjust database**

**create user software identified by password;**

**// we should give privilege to user to connect**

**alter user hr account unlock;**

**//to connect as admin**

**connecgt system**

**alter user hr identified by password2 ;**

**grant connect to software;**

**// give user privilege to create**

**grant resource to software;**

**// in column level references means foreign key**

**// in table level we must use foreign key keyword**

**// at start**

**Connect software**

**Password**

**// A DDL or DCL statement executes (automatic commit).**

* **We can create save point inside our work**
* **SAVEPOINT update\_done**
* **To return to this save point**
* **ROLLBACK TO update\_done**
* **BEGIN TRANSACTION/END TRANSACTION to make tansaction baundary**
* **Exec Product\_Line\_sale to execute funcion or precedure**

Set operators 🡪 the expressions in the SELECT lists must match in number and data type.

* **Union 🡪 get both queries without duplication**
* **SELECT employee\_id, job\_id FROM employees**

**UNION**

**SELECT employee\_id, job\_id FROM job\_history;**

* **Union All 🡪 get both queries with duplication**
* **SELECT employee\_id, job\_id, department\_id**

**FROM employees**

**UNION ALL**

**SELECT employee\_id, job\_id, department\_id**

**FROM job\_history ORDER BY employee\_id;**

* **intersection 🡪 return data in both queries**

**SELECT employee\_id, job\_id FROM employees**

**INTERSECT**

**SELECT employee\_id, job\_id FROM job\_history;**

* **minus 🡪 rows of first query without the second one**
* **SELECT employee\_id,job\_id**

**FROM employees**

**MINUS**

**SELECT employee\_id,job\_id**

**FROM job\_history;**

* **create trigger t\_name**

**{before/after/instead of}{insert/delete/update} on t\_name**

**[for each {row/statement} ][when condtion]**

**< trigger sql here>**

* **creating view**

**CREATE VIEW EXPENSIVE\_STUFF\_V AS**

**SELECT PRODUCT\_ID, PRODUCT\_NAME, UNIT\_PRICE**

**FROM PRODUCT\_T**

**WHERE UNIT\_PRICE >300**

**WITH CHECK\_OPTION;**