Database Management System

Introduction

Concept of Database

Database: It is a collection of interrelated data files/tables.

Table: It is collection of similar records.

Record: It is collection of meaningful attribute values.

Attribute: Property of an entity.

Entity: Areal world object which should be distinguished from others.

Table 1. Sample table(relation/entity set)

Sid	Sname	Marks
S1	Ram	30
S2	Mohan	40
S3	Sita	50
S4	Ravan	0

The table consists of rows and columns. Rows are called records or tuple or entity. Column is called attribute or field.

DBMS Database Management system is a software package which enables user to create and maintain a database .

The DBMS is hence a general-purpose software system that facilitates the process of defining, constructing and manipulating databases for various applications.

E.g. Oracle, Ingress, Sybase, Dbase 3+, Foxbase, Foxpro, Ms access, Database, Dataflex, SQL Server etc.

Data, information and knowledge

Data

Data is/are the facts of the World. For example, take yourself. You may be 5ft tall, have brown hair and blue eyes. All of this is “data”. You have brown hair whether this is written down somewhere or not.

In many ways, data can be thought of as a description of the World. We can perceive this data with our senses, and then the brain can process this.

Human beings have used data as long as we’ve existed to form knowledge of the world.

Until we started using information, all we could use was data directly. If you wanted to know how tall I was, you would have to come and look at me. Our knowledge was limited by our direct experiences.

Information

Information allows us to expand our knowledge beyond the range of our senses. We can capture data in information, then move it about so that other people can access it at different times.

Here is a simple analogy for you.

If I take a picture of you, the photograph is information. But what you look like is data.

I can move the photo of you around, send it to other people via e-mail etc. However, I'm not actually moving you around – or what you look like. I'm simply allowing other people who can't directly see you from where they are to know what you look like. If I lose or destroy the photo, this doesn't change how you look.

CHARACTERISTICS OF VALUABLE INFORMATION.

In order for information to be valuable it must have the following characteristics, as adapted from Ralph M. Stair's book, Principles of Information Systems:

Accurate. Accurate information is free from error.

Complete. Complete information contains all of the important facts.

Economical. Information should be relatively inexpensive to produce.

Flexible. Flexible information can be used for a variety of purposes, not just one.

Reliable. Reliable information is dependable information.

Relevant. Relevant information is important to the decision-maker.

Simple. Information should be simple to find and understand.

Timely. Timely information is readily available when needed.

Verifiable. Verifiable information can be checked to make sure it is accurate.

In Brief

Data: Facts, a description of the World

Information: Captured Data and Knowledge

Knowledge: Our personal map/model of the World

Data processing Vs data management

Data processing

Data processing refers to the process of performing specific operations on a set of data or a database. A database is an organized collection of facts and information, such as records on employees, inventory, customers, and potential customers. As these examples suggest, numerous forms of data processing exist and serve diverse applications in the business setting.

Data processing primarily is performed on information systems, a broad concept that encompasses computer systems and related devices. At its core, an information system consists of input, processing, and output. In addition, an information system provides for feedback from output to input. The input mechanism (such as a keyboard, scanner, microphone, or camera) gathers and captures raw data and can be either manual or automated. Processing, which also can be accomplished manually or automatically, involves transforming the data into useful outputs. This can involve making comparisons, taking alternative actions, and storing data for future use. Output typically takes the form of reports and documents that are used by managers. Feedback is utilized to make necessary adjustments to the input and processing stages of the information system.

The processing stage is where management typically exerts the greatest control over data. It also is the point at which management can derive the most value from data, assuming that powerful processing tools are available to obtain the intended results. The most frequent processing procedures available to management are basic activities such as segregating numbers into relevant groups, aggregating them, taking ratios, plotting, and making tables. The goal of these processing activities is to turn a vast collection of facts into meaningful nuggets of information that can then be used for informed decision making, corporate strategy, and other managerial functions.

Data Management

The official definition provided by DAMA International, the professional organization for those in the data management profession, is: "Data Resource Management is the development and execution of architectures, policies, practices and procedures that properly manage the full data lifecycle needs of an enterprise." {{DAMA International}} This definition is fairly broad and encompasses a number of professions which may not have direct technical contact with lower-level aspects of data management, such as relational database management.

Alternatively, the definition provided in the DAMA Data Management Body of Knowledge (DAMA-DMBOK) is: "Data management is the development, execution and supervision of plans, policies, programs and practices that control, protect, deliver and enhance the value of data and information assets."

The concept of "Data Management" arose in the 1980s as technology moved from sequential processing (first cards, then tape) to random access processing. Since it was now technically possible to store a single fact in a single place and access that using random access disk, those suggesting that "Data Management" was more important than "Process Management" used arguments such as "a customer's home address is stored in 75 (or some other large number) places in our computer systems." During this period, random access processing was not competitively fast, so those suggesting "Process Management" was more important than "Data Management" used batch processing time as their primary argument. As applications moved more and more into real-time, interactive applications, it became obvious to most practitioners that both management processes were important. If the data was not well defined, the data would be mis-used in applications. If the process wasn't well defined, it was impossible to meet user needs.

Purpose of Database system

Database System=Database + DBMS

In early days, database applications ware built on top of file system.

Following are the drawback of using file system to store data which can be overcome by database system.

- Data redundancy and inconsistency.
  - Duplication of same information at several places are possible.
  - All copies may not be updated properly.
- Difficulty in accessing data
  - May have to write a new application program to satisfy an unusual request.
  - E.g. Find all students with same marks.
  - could generate this data manually, but a tedious job.
- Data Isolation
  - Data in different files.
  - Data in different formats.
  - Difficult to write new application programs.
- Multiple users
  - Want concurrency for faster response time
  - Need protection for concurrent updates.
  - E.g. two customers depositing funds in the same account at the same time.
- Security problems
  - Every user of the system should be able to access only the data they are permitted to see.
  - Difficult to inforce this with application programs.
- Integrity problems
  - Data may be required to satisfy constraints.
  - E.g. No account balance should be below Rs 500.
  - Again difficult to enforce or to change constraints with the file processing approach.
- Atomicity of updates
  - Failures may leave database in an inconsistent state with partial update carried out.
  - E.g. Transfer of funds from one account to another should either complete or not happen at all.

Functionality of a database system

- Specifying the database structure
  - data definition language
- Manipulation of database
  - Query processing and query optimisation.
- Integrity enforcement
  - integrity constraints
- Concurrent control
  - multiple user environment.
- Crash recovery
- Security and authorization.

Types of DBMS

Several criteria can be used to classify DBMSs. Following are the criteria and types of DBMS according to them.

1. Data Model
   1.1. Relational model
   1.2. Object data model
   1.3. Object-relational model
   1.4. Network model
   1.5. Hierarchical model

Many current DBMSs use the relational data model or object data model. Many legacy applications still run on database systems based on hierarchical and network data models. Relational DBMSs are extending their models to incorporate object based concepts and other capabilities. These systems are referred to as object-relational systems.

2. Number of users
   2.1. Single user systems
   2.2. Multiuser systems
3. Number of sites
   3.1. Centralized DBMS
   Data is stored at a single computer site.
   3.2. Distributed DBMS

can have the actual database and DBMS software distributed over many sites, connected by a computer network.

3.3. Federated DBMS
Participating database and DBMSs are heterogeneous. It's a combination of centralized and distributed DBMS.

4. Cost
5. Type of access path
6. Purpose
   6.1. General-purpose DBMS
   6.2. Special-purpose DBMS
   E.g. DBMS for airline reservation system.

DBMS Architecture

Database systems are usually partitioned into two or three parts as in fig 1.

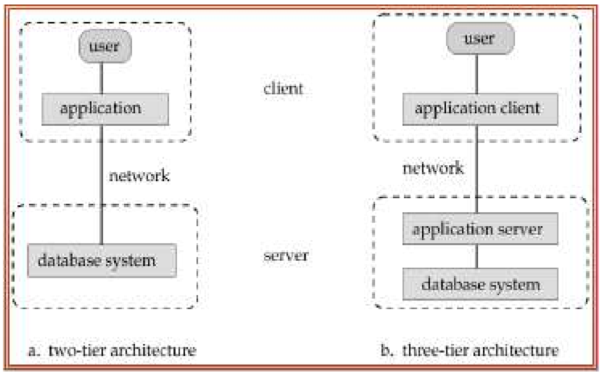


Figure 1 two-tier and three-tier architecture

In a two-tier architecture, the application is partitioned into a component that resides at the client machine, which invokes database system functionality at the server machine through query language statements.

In a three-tier architecture, the client machine acts as merely a front end and does not any direct database calls. instead , the client end communicates with an application server, usually through a forms interface. The application server in turn communicates with a database system to access data. The business logic of the application, which says what actions to carry out under what conditions, is embedded in the application server, instead of being distributed across multiple client. Three-tier applications are more appropriate for large applications and for the applications that run on World Wide Web.

Views of Data

A major purpose of a database system is to provide users with an abstract view of data. That is the system hides certain detail of how the data are stored and maintained.

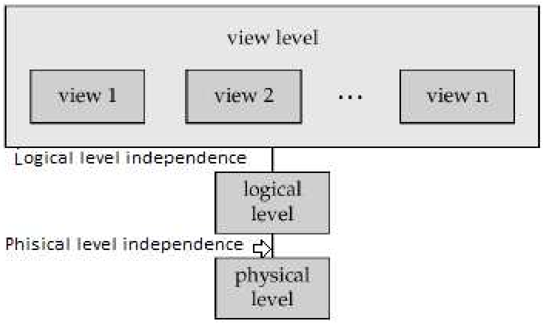


Figure 2 three levels of data abstraction or views

Physical level: How the data are actually stored.

Logical level: What data are stored in database and what relationship exist among those data. Thus, the logical level describes the entire database in terms of small number of relatively simple structures.

View level: describes only a part of entire database. The system may provide many views of the same database.

Data Independency

The ability to modify a scheme definition in one level without affecting a scheme definition in a higher level is called data independence.

There are two kinds of data independency

1. Physical data independence

- The ability to modify the physical scheme without causing logical schema to be modified.
- Modifications at this level are usually to improve performance.

2. Logical data independence
   - The ability to modify conceptual schema without causing any modification on view level.
   - Usually done when logical structure of database is altered.

Data Models

These are the model that determines the logical structure of a database and fundamentally determines in which manner data can be stored, organized, and manipulated. Most popular data model for database design is relational model.

Following are some common data models.

1. Flat file model
2. Network data model
3. Hierarchical data model
4. Entity Relationship model
5. Relational data model
6. Object data model
7. Object-Relational data model

Flat file model: The database is a collection of flat files. Data are stored in these files where didn't think about relationships. These relationships ware present but they were not named or maintained.

Network data model: The network database is a collection of set occurrences. The two basic data structuring concept are records and sets. A set occurrence will have one owner record and many member records.

Hierarchical model: The hierarchical database is a collection of tree occurrences. The two data structuring concepts are record and parent child relationship (PCR). An occurrence of the PCR type consists of one record of the parent record-type and a number of record of child record type.

A hierarchical database schema consists of a number of hierarchical schemas. Each hierarchy consists of a number of record type and PCR types.

Entity-Relationship model: This is just a model in which we represent entity sets and their relationships through graphical diagrams. Entity-sets are same as table or record type. We will see this model in detail in next unit where we will come across different notations and their plotting.

Relational data model: The relational database is a collection of tables. These tables will be related to each other with the help of foreign keys.

Object data model: The database is a collection of objects. The relationship among the objects is maintained using the foreign key as attribute of the class built for record type. The object data model has an advantage of being able to store data like picture, sound files or videos etc.

Object Relational data model: The relational data model is extended to incorporate the features of object data model so that picture or such unstructured data could be stored with the facilities of relational environment.

Data dictionary

The data dictionary is considered to be a special type of table, which can only be accessed and updated by database system itself (not a regular user). A database system consults the data dictionary before reading modifying actual data.

The output of DDL is placed in the data dictionary, which contains metadata, which is data about data. Following are the data which must be present in the data dictionary.

- Name of the relation.
- Name of the attribute of each relation
- Domains and lengths of attributes.
- Name of views defined on the database, and definition of those views.
- Integrity constraints.
- Name of authorized users.
- Authorization and accounting information about users.
- Password or other information used to authenticate users.
- Number of tuples in each relation.
- Method of storage in each relation.

May also note the storage information (sequential hash or heap) of relation and the relation where each relation is stored.

May also store following.

- Name of index.
- Name of the raltion being indexed.
- Attribute on which the indexing is done.
- Type of index formed.

SQL (Structured Query Language)

Introduction

The history of SQL begins in an IBM laboratory in San Jose, California, where SQL was developed in the late 1970s. The initials stand for Structured Query Language, and the language itself is often referred to as "sequel." It was originally developed for IBM's DB2 product (a relational database management system, or RDBMS, that can still be bought today for various platforms and environments). In fact, SQL makes an RDBMS possible. SQL is a nonprocedural language, in contrast to the procedural or third generation languages (3GLs) such as COBOL and C that had been created up to that time. *Nonprocedural* means *what* rather than *how*. For example, SQL describes what data to retrieve, delete, or insert, rather than how to perform the operation.

Types of language

Data Definition Language (DDL) statements are used to define the database structure or schema. Some examples:

CREATE - to create objects in the database.
ALTER – alters structure of the database.
DROP - delete objects from the database
TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed.
COMMENT - add comments to the data dictionary.
RENAME - rename an object.

Data Manipulation Language (DML) statements are used for managing data within schema objects.
Some examples:
SELECT - Retrieve data from the database
INSERT - Insert data into a table.
UPDATE - Updates existing data within a table.
DELETE - deletes all or selected records from a table, the space for the records remain.
MERGE - UPSERT operation (insert or update)
CALL - Call a PL/SQL or Java subprogram.
EXPLAIN PLAN - explain access path to data.
LOCK TABLE - control concurrency.

Data Control Language (DCL) statements are used to manage the users authority.
Some examples:
GRANT - gives user's access privileges to database.
REVOKE - withdraw access privileges given with the GRANT command.

Transaction Control (TCL) statements are used to manage the changes made by DML statements. It allows statements to be grouped together into logical transactions.
Some examples:
COMMIT - save work done.
SAVEPOINT - identify a point in a transaction to which you can later roll back.
ROLLBACK - restore database to original since the last COMMIT.
SET TRANSACTION - Change transaction options like isolation level and what rollback segment to use.

Creating a table

SQL provides CREATE TABLE command using which we can define structure of a table. Each table column definition is a single clause in the create table syntax which is separated from each other by a comma. Finally, the SQL statement is terminated with a semi colon.

Syntax:

```
CREATE TABLE <TableName>
(
        <attribute1> <datatype>(<size>),
        <attribute2> <datatype>(<size>),
        :
        .
        <attributen> <datatype>(<size>)
);
```

Rule for creating table:

- A name can have maximum up to 30 characters.
- Alphabets from A-Z, a-z and number from 0-9 are allowed
- A name should begin with an alphabet.
- The use of special characters like _ is allowed and also recommended. (Special characters like $, # are allowed only in Oracle).
- SQL reserved words not allowed.

Data types in Oracle

Following are some most popular data types. Using these data types we can define the domain of a attribute.

Number (precision, scale) The Number data type is used to store numbers (fixed or floating point). Maximum size is 38 digits of precision.

Char(size) This data type is used to store character string values of fixed size. The size in brackets determines the number of characters the cell can hold. If the inserted string has less than that char then rest of the entry is padded with space. The maximum this data type can hold is 2000B.

Varchar(size) This data type is used to store variable length alphanumeric data. the inserted values will not be padded by spaces. The maximum this data type can hold is 4000B.

Nchar(size) This data type is similar to char except the fact that it can store any natural language character. It takes 3B to store one char.

Nvarchar(size) This data type is similar to char except the fact that it can store any natural language character. It takes 3B to store one char.

Date This data type is used to represent date and time. The standard format is DD-MMM-YY as in 23-JAN-14.

Other data types are available in oracle like long, raw etc.

Example: Create a table Student with following structure.
Student(sid,sname,saddr,marks).

Sol:

```
create table Student
(
        sid varchar(5) PRIMARY KEY,
        sname varchar(15),
        saddr varchar(30),
        marks number(3)
);
```

This statement can be written in a single line. This statement when executed will create a table with name Student having four attributes sid, sname, saddr and marks. The key word PRYMARY KEY has been used to enforce a constraint which will not allow entries in sid which are repeated .And at the same time the entries cannot be left blank.

Inserting values in table

The table created should be loaded with data to be manipulated latter.

Syntax:

INSERT INTO <tablename>[(list of attribute)] values(v1,v2,v3,...,vn);

v_i represents the values for corresponding attributes.

Example:

insert into Student(sid,sname,saddr,marks) values(‘s1’,’ram’,’bilaspur’,60);

The above command will operate in following two stape.

1. Creates a new row(empty) in the database table

2. Loads the values passed (by the insert statement) into the columns specified.

Note: Character values places within the insert into statement must be enclosed in single quotes (‘).

If there are less values being described than there are columns in the table then it is mandatory to indicate both the table column name and its corresponding value in the insert into statement.

Retrieving data from table

The SELECT command is used to retrieve the records selected from one or more tables. Following are the possibility of viewing (retrieving) data from a table.

All rows and all columns

Syntax: Select * from <table_name>;

Example: Select * from Student;

Result:

Sid	sname	saddr	marks
S1	ram	bilaspur	60
S2	mohan	raipur	40
:			
Sn	ramesh	puri	80

Oracle allows a meta character asterisk (*) to mean all attributes of the table.

Filtering table data

While viewing data from a table, it is rare that all the data from the table will be required each time. SQL provides a method of filtering table data which are following.

- All rows of selected columns.
- Selected rows of all columns.
- Selected rows of selected columns.

<u>All rows of selected columns</u>

The retrieval of specified columns can be done using following syntax.

SELECT column1,column2,...,$column_k$ form <table name>;

Example: Retrive the sid and marks of all the student.

SELECT sid, marks from Student;

sid	marks
S1	60
S2	40
:	
Sn	80

<u>Selected rows of all columns</u>

If we have to retrieve selected records we will have to specify selection condition. Following is the syntax.

SELECT * from <table_name> WHERE <condition>;

Example: Retrieve the records of those students who have marks greater than 50.

SELECT * from Student WHERE marks>50;

Result:

Sid	sname	saddr	marks
S1	ram	bilaspur	60
S4	gajab	raipur	70
:			
Sn	ramesh	puri	80

only those records which has marks greater than 50.

Condition is following format.

<Attribute_name > <operator> <attribute_name/value>

Means condition will have a logical expression which will evaluate either to TRUE or FALSE. The records for which the condition will evaluate to TRUE those records will be selected.

Selected rows of selected column

To view a specific set of rows and column we will use following syntax.

SELECT <List of column> From <table_name> WHERE <condition>;

Example: Retrieve the sid and marks of those students who have got more than 50 marks.

SELECT sid, marks FROM Student WHERE marks>50;

Result:

Sid	marks
S1	60
S4	70
:	
Sn	80

Only the sid and marks of those students who have more than 50 marks.

Eliminating Duplicate rows while using a select statement

SELECT DISTINCT <attribute list> FROM <table_name>;

DISTINCT is a keyword used to eliminate the duplicate rows.

example: Retrieve the different marks given to the students.

SELECT DISTINCT marks FROM Student;

Result:

marks
60
40
30
:
80

Sorting data in a table

Oracle allows data from a table to be viewed in sorted order. Following is the syntax.

SELECT * FROM <table_name> ORDER BY <attribute1>,<attribute2>[order];

Example: Select * from Student order by marks DESC;

By default order is ASC which stands for ascending order. For viewing data in descending order the word DESC must be mentioned after the column name.

Creating a table from a table

Syntax: CREATE TABLE <table_name>(<attribute1>,<attribute2>) AS SELECT <attribute1>,<attribute2> FROM <table_name>;

Example: Create table chhotastudent(sid,marks) AS select sid, marks from student;

When the above statement will be executed a table named chhotastudent will be created with two attributes sid and marks. The data types of attributes will be taken from the student table. The result will a table named chhotastudent with two attributes and the values of those attributes for all the records from table student.

To create a target table without the records from the source table (i.e. create the structure only), the select statement must have a where clause. The where clause must specify a condition that cannot be satisfied.

Inserting data into table from another table

Syntax: Insert into <table_name> Select <attribute1>, <attributen>from <table_name>;

Example: Insert into Chhotastudent Select sid, marks from Student;

Insertion of a data set into a table from another table

Syntax:
Insert into <table_name> select <list of attributes> from <table_name> where <condition>;
Example: Insert into Chhotastudent Select sid, marks from Student where marks<=40;

Delete Operations

The DELETE command deletes rows from the table that satisfies the condition provided by its where clause, and return the number of record deleted.

Note: If the DELETE statement is executed without where clause then all the rows are deleted.

Removal of all Rows

Syntax: DELETE FROM <table_name>;
Example: Empty the Student table.
Delete from student;

Removal of Specified Rows

Syntax: DELETE FROM <table_name> where <condition>;
Example: Delete records of those students who have less than 10 marks.
DELETE from student where marks<10;

Updating the content of a table

The UPDATE command is used to change or modify the content of existing records of a table.

Updating all rows

Syntax: UPDATE <table_name> SET <attribute1>=<expression1>, <attributen>=<expression>;

Example: Update the saddr of student by changing its city name to bilaspur;

UPDATE Student SET saddr='bilaspur';

Updating a selected set of records in a table

Syntax: UPDATE <table name> SET <attribute>=<expression> WHERE <condition>;

Example: Update the marks of those students whose marks is less than 10 and set to 0.

Update student set marks=0 where marks<10;

Modifying the structure of tables

The ALTER TABLE command is used to modify the structure of a table. With ALTER TABLE it is possible to add or delete columns, or change the data type of existing columns.

ALTER TABLE works by making a temporary copy of the original table. The alteration is performed on the copy, them the original table is deleted and the new one is renamed. While ALTER TABLE is executing the original table is still readable by users of ORACLE.

Syntax for adding new column:

ALTER TABLE <table_name> ADD(<NewColumnName> <data type(size)>, <NewColumnName> <data type(size)>);

Example: Add a new column deptno to the table Student whose data type should be varchar(3).

ALTER TABLE Student ADD(deptno varchar(3));

Syntax for dropping a column from a table

ALTER TABLE <table_name> DROP COLUMN <columnName>;

Example: Drop the column saddr from the table student.

ALTER TABLE Student DROP COLUMN saddr;

Syntax for modifying existing columns

ALTER TABLE <table_name> MODIFY(<columnName> <NEWDATATYPE>(<newsize>));

Example: Alter the Student table to allow the sname field to hold maximum of 40 characters.

ALTER TABLE Student MODIFY (sname varchar(40));

Restrictions on the ALTER TABLE

The following tasks cannot be performed when using the ALTER TABLE command.

- Change the name of a table.
- Change the name of the column
- Decrease the size of a column if table data exists.

Renaming tables

Oracle allows renaming of tables. The syntax is as follows.

RENAME <table_name> TO <new_table_name>;

Example: Change the name of the table Student to univstudent.

RENAME Student to univstudent;

Destroying Tables

DROP TABLE statement with the table name can destroy a specific table. If a table is dropped all records held within it are lost and cannot be recovered.

Syntax: DROP TABLE <table_name>;

Example: Remove the table chhotastudent along with the data held.

DROP TABLE chhotastudent;

Truncating tables

TRUNCATE TABLE empties a table completely. It is equivalent to a DELETE statement that deletes all rows, but there are practical differences .

TRUNCATE TABLE differs from DELETE in the following ways:

- Truncate operation drop and re-create the table, which is much faster than deleting rows one by one.
- Truncate operations are not transaction-safe(i.e. an error will occur if an active transaction or an active table lock exists)
- The number of deleted rows are not returned.

Syntax: TRUNCATE TABLE <table_name>;

Example: Truncate the table Student.

TRUNCATE TABLE Student;

Displaying the table Structure

To display information about the columns defined in a table use the following Syntax.

DESCRIBE <table_name>;

This command displays the column names, whether NULL values are allowed or notand the data type with size.

Example: Show the structure of table Student.

DESCRIBE Student;

Note : In place of using DESCRIBE the first four characters can be used to serve the same purpose. For example

DESC Student;

Data constraints

Business rules, which are being enforced on data being stored in a table, are called *constraint*. Constraints super control the data being entered into a table for permanent storage.

Oracle permits data constraints to be attached to table columns via SQL syntax that checks data for integrity prior storage. Even if a single column of the record being inserted into the table fails a constraint, the entire record is rejected and not stored in the table.

Both the CREATE TABLE and ALTER TABLE SQL commands can be used to attach constraints to a table column.

Types of data constraints

There are two types of data constraints namely I/O constraints and business rule constraints.

I/O Constraints

This data constraint determines the speed at which data can be inserted or extracted from a Oracle table.

a) PRIMARY KEY

A primary key column in a table has special attributes:

- It defines the column as mandatory column (i.e. the column can not be left blank).
- The data present throughout the column must be unique.

At column level

Syntax:

<column name> <data type(size)> PRIMARY KEY

At table level

Syntax:

PRIMARY KEY(column name[,column name])

b) FOREIGN KEY

Foreign key represents relationship between tables. A foreign key is a column (or a group of columns) whose values whose values are derived from the primary key or unique key of some other table or same table.

At table level

Syntax:

<attribute name> <data type> REFERENCES <table name>[(<attribute>)]

At table level

Syntax:

Foreign key (<columnName>[,<ColumnName>]) References <TableName>[(<columnName>[,<Column Name>])]

c) UNIQUE

The Unique column constraint will not allow duplicate values however NULL is allowed.

At column level

Syntax:

<columnName> <data type(size)> UNIQUE

At table level

Syntax:

Unique (<columnName>[,<columnName>])

d) NOT NULL

A NULL value is different from a blank or zero. A NULL is a unknown or not existing value.

At column level

Syntax:

<columnName> <data type(size)> NOT NULL

At table level

NOT NULL constraint **cannot** be applied at table level

Business rule constraints

Business rule are determined by business managers which will vary from system to system

a) CHECK

Check constraints are used to validate business rules. The check constraint will have a logical expression that evaluates either to TRUE or FALSE.
A check constraint takes subsequently longer time to execute as compared to other above constraints.

Syntax at column level
<ColumnName> <data type>(size) CHECK (<logical expression>)

Syntax at table level
CHECK(<logical expression>)

General Form of SELECT statement

4SELECT [DISTINCT]<LIST OF ATTRIBUTE>
1FROM <LIST OF TABLE NAME>
2[WHERE <condition>]
3[GROUP BY <attributename>[HAVING <condition>]]
5[ORDER BY<attribute>[DESC]];

The numbers in front of the statement reflect the order of execution.
We are mostly aware of all the above clauses except the Group By clause.

Group By clause

The Group By clause is used to retrieve the information group wise. The group by clause makes one record for each group. From each group the attribute on which it has been grouped, can be projected as it is and the rest attributes can be projected only with the help of aggregate functions.

Aggregate Function

Aggregate functions are those functions which are used on group of values. Examples of aggregate functions are following.

MIN(<attribute>)

Finds the minimum value in an attribute.

MAX(<attribute>)

Finds the maximum value in an attribute.

SUM([Distinct]<attribute>)

Finds the sum of all the values of attribute.

COUNT([distinct]<attribute>)

Counts the number of records in the result or the attribute.

AVG([Distinct]<attribute>)

calculate the average of the values in the attribute.

Significance of Distinct.

When Distinct is used then the duplicate values are considered only once.

Nested Query

When a query is written inside another query then it is called Nested query. Anywhere in query we can have sub query. The inner query is also called sub query.

R

A
1
2
10
4

S

B
1
5
10
11

Example: Retrieve the value of attribute A from relation R which is also present in attribute B of relation S.

Select R.A from R where R.A **IN** (Select S.B from S);

Operators used in nested query

Following operators can be used in nested queries.

IN, NOT IN

The IN operator will take a single value in LHS and a set of values in RHS. The IN operator will evaluate to true only if The LHS is present in the RHS set of values. NOT IN is just opposite of IN operator.

OP ANY

OP stands for operator. Operators can be =, <>, >, <, >=,<=. This operator also takes a single value in LHS and a set of value in RHS. The condition evaluates to true if the value on the left hand side satisfies the operator condition with any of the values in the set.

OP ALL

OP is similar to above. This operator evaluates to true if it satisfies the operator condition for all the values of the set.

EXISTS, NOT EXISTS

This operator only a SQL query on RHS. This evaluates to true if the result of query is having at least one record. It evaluates to False if there is no record in the result of query following the EXIST operator. NOT EXISTS is just opposite of EXISTS.

Correlated Nested Query

A query is said to be a correlated nested query if the table listed in outer query is also used in inner query.

Example:

Select R.A from R where EXISTS(Select S.B from S where R.A=S.B);

Substring comparison

LIKE operator is used for substring comparison.

Example: Retrieve the name of students whose address starts from 'Bilas'.

Sol: SELECT SNAME FROM STUDENT WHERE SADDRESS LIKE 'Bilas%';

% is used for representing any number of character.

_(underscore) is used to represent a single character.

Arithmetic operators

Arithmetic operators can be used in select clause and also in where clause. Operators are +,-,*,/ for numeric attribute and '||' for string.

Example : Select name, 1.1*salary from EMP.

BETWEEN operator.

Can be used to mention a condition for a range.

Example: Select * from EMP where salary BETWEEN 30000 AND 40000;

BETWEEN is always used with AND operator.

IS operator

IS operator is used to compare NULL. NULL cannot be compared using '=' operator.

Example: Retrieve the name of those students who don't have a passport number.

Sol: Select sname from student where ppno **IS** null;

ppno is a attribute of table student which stores the passport number.

Views in SQL

View is a virtual table which is derived from other tables. It is based on some SQL query.

Example: Create a view faculty_view on table student which will have only attributes sid, sname and marks from table student.

Sol: Create view faculty_view AS select sid, sname, marks from student;

Relational Algebra

- by default eliminates duplicates from the result.

Basic Operators	Derived Operators
π = Projection	⋈ : Join
σ = Selection	∩ : Intersection
X : Cross Product	R∩S = R − (R−S)
U : Union	/ : division
− : Set difference	
ρ : Rename	

A basic set of relational model operations constitute the relational algebra. These operations enable the user to specify basic retrieval requests. The result of retrival 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

π Projection (Projecting collumns)

Syntax: $\pi_{attributes}(R)$

R

A	B	C
1	1	2
1	1	3
2	1	1

$\pi_{AB}(R)$ =

A	B
1	1
1	1
2	1

⇒

A	B
1	1
2	1

.. NO commutativity

R

A	B	C
1	1	2
2	1	1

} m

S

B	C	D
1	1	2
2	2	1
2	1	2

n {

R x S

A	B	C	B	C	D
1	1	2	1	1	2
1	1	2	2	2	1
1	1	2	2	1	2
2	1	1	1	1	2
2	1	1	2	2	1
2	1	1	2	1	2

} m×n

Purpose: to compare one table data to other table data.

σ Selection (Selecting rows)

Syntax

$\sigma_P(R)$ - select tuples from relation R, satisfying condition P.

Eg. $\sigma_{C>2}(R)$ =

A	B	C
1	1	3

→ String → =, ≠, SUBSTRING_OF

→ Degree is same as R.

$|\sigma_c(R)| \leq |R|$

→ A sequence of select can be applied in any order.

→ we can combine a cascade of select op. using AND.

X Cross Product

Syntax,

R x S → All the attributes of R followed by attributes of S with all combination of tuples R&S.

Join ⋈

Cross product followed by selection and Projection.

conditional join $⋈_c$

- we need to specify the condition of join conditions - <, >, <=, >= , =, ≠

$R ⋈_c R = \pi_{ABCBD}(\sigma_c(R \times S))$

eg

$R ⋈_{R.C>S.B} S = \pi_{ABCBCD}(\sigma_{R.C>S.B}(R \times S))$

Natural Join ⋈ ×

Checks for equality between common fields.

$R \bowtie S = \pi \left(\sigma_{R.B=S.B \text{ AND } R.C=S.C} (R \times S) \right)$

Distinct collumn.

A	B	C	D

minimum tuple – 0

(ABC) ⋈ (DE) = (ABC) × (DE)

No common attribute

Left Outer Join ⟕

R ⟕ S = R ⋈ S and Tuples of left side relation those failed join condition.

A	B	C	D
1	1	2	4
2	1	2	4
1	2	1	NULL

→ Minimum tuples → No of tuples in R.

Right Outer Join ⟖

R ⟖ S = R ⋈ S + Tuples of Right

A	B	C	D
1	1	2	4
2	1	2	4
NULL	3	2	3
NULL	4	1	6

Full outer Join ⟗

R ⟗ S = R ⟕ S ∪ R ⟖ S

A	B	C	D
1	1	2	4
2	1	2	4
1	2	1	NULL
NULL	3	2	3
NULL	4	1	6

Set Operators.

[∪, ∩, −]

Union Compitability

Two relations $R(A_1, A_2, A_3, \ldots A_n)$ and $S(B_1, B_2, B_3, \ldots, B_n)$ are said to be union compitible if they have the same degree 'n' and $dom(A_i) = dom(B_i)$ for $1 \le i \le n$.

∪ Union, ∩ - Intersection,

student

FN	LN

✓ Instructors

FNAME	LNAME

− - set difference –

- A relation (table) is a set of records.

→ Union and intersection are commutative.

→ Set difference is not commutative.

Division Operator

Enrolled

Sid	Cid
S1	C1
S1	C2
S2	C1
S2	C2
S1	C3
S3	C3

Course

cid
C1
C2
C3

→ Retrive Sids who are enrolled some or atleast 1 course.

π_{sid} (Enrolled) =

Sid
S1
S2
S3

→ Retrive sids who enrolled all the courses

$= \pi_{sid,cid}(\text{Enrolled}) \div \pi_{cid}(\text{course})$

"÷" Devision operator is derived from π, −, ×.

$A(x, y) / B(y) = \pi_x(A) - \pi_x(\pi_x A \times B - A)$

Q. Sample Database

Suppliers (Sid, Sname, rating)
Parts (Pid, Pname, color)
catalog (Sid, pid, cost)

Q1. Retrieve Sids of the Supliers whose rating is greater than 7.

$\Rightarrow \pi_{sid}(\sigma_{rating>7}(Suppliers))$

Q2. Retrieve Sid of Suppliers who suply some red parts.

1. $\pi_{sid}(\sigma_{col=Red}(Catalog \bowtie Parts))$
2. $\pi_{sid}(\sigma_{col=Red}(catalog \bowtie (\sigma_{col=Red}(PARTS))))$
3. $\pi_{sid}(\pi_{Pid}(\sigma_{col=Red}(Parts)) \bowtie \pi_{Sid\,Pid}(catalog))$

Q3. Retrieve Snames of the Suppliers who suply some red parts.

i) $\pi_{Sname}(\sigma_{col=Red}((catalog \bowtie PARTS) \bowtie suppliers))$

ii) $\pi_{Sname}(\pi_{sid}(\pi_{Pid}(\sigma_{col=Red}(Parts)) \bowtie \pi_{Sid\,Pid}(catalog)) \bowtie \pi_{sid\,Sname}(Suppliers))$

Q4 Retrieve Sids of the Suppliers who supply some red part OR some green part.

i) $\pi_{sid}(\sigma_{col=Red \lor col=Green}(Parts) \bowtie Catalog)$

ii) $T_1 \leftarrow \pi_{sid}(\sigma_{col=Red}(Parts) \bowtie catalog)$

$T_2 \leftarrow \pi_{sid}(\sigma_{col=green}(Parts) \bowtie catalog)$

$T_1 \cup T_2$

Q5 Retrive Sids of Suppliers who supplied atleast one parts.

$\pi_{sid}(catalog)$

Q6 Retrive Sids of the suppliers who supplied atleast two parts.

$\rho(T_1, catalog)$

$\rho(T_2, catalog)$

$\pi_{T_1.sid}(\sigma_{T_1.Sid = T_2.Sid \land T_1.Pid \neq T_2.Pid}(T_1 \times T_2))$

Q. collect 3

Tuple Relational Calculus

Relational calculus is a formal query language where we write one declarative expression to specify a retrieval request and hence there is no discription of how to evaluate a query; a calculus expression specifies what is to be retrieved rather than how to retrieve it. Therefore the relational calculus is considered to be a nonprocedural language.

This differs from relational algebra, where we must write a sequence of operations to specify a retrieval request; hence it can be considered as a procedural way of stating a query.

The expressive power of the two languages is identical.

A simple tuple relational calculus query is of the form

$\{t \mid COND(t)\}$

t is a tuple variable and COND(t) is a conditional expression involving t.

results set of all tuples t that satisfy condition COND(t).

eg

$\{t \mid Emp(t)$ and $t.Salary > 5000\}$

Find all emp whose salary is greater than 5000.

Formulla.

$\{t_1.A_1, t_2.A_2, \ldots t_n.A_n \mid cond(t_1, t_2, \ldots t_n, t_{n+1}, \ldots t_{n+m})\}$

A_i is the attribute of relation on which t_i ranges. and cond is a condition or well formed formulla.

A formula is made up of the predicate calculus atoms, which can be one of the following.

1. $R(t_i)$ R is the name of relation, t_i is the tuple variable or $t_i \in R$.
2. $t_i.A$ OP $t_j.B$
3. $t_i.A$ OP C or C OP $t_j.B$

OP is one of $\{=, >, \geq, <, \leq, \neq\}$

i) Every atom is a formula.

ii) If F_1 & F_2 are formula then so are $F_1 \wedge F_2$, $F_1 \vee F_2$, $\neg F_1$, $\neg F_2$

iii) If F is formula $\exists t (F)$ is so.

iv) ,, $\forall t (F)$,,

Q. Retrive sname of the suppliers

R.A : π_{sname} (suppliers)

TRC: $\{t.sname \mid t \in Suppliers\}$

Q. SID of supplier whose rating is greater the 7.

TRC: $\{t.sid \mid t \in Supplier \wedge t.rating > 7\}$

(31)

Domain Relational Calculus

It is another type of relational calculus also called Domain calculus.

It differs from TRC in the type of variable used in formula: rather than having variables range over tuples, the variables ranges over single value from the domain of attribute. To form a relation of degree n for a query result, we must have n of these domain variables – one for each attribute.

Expression is of form:

$\{x_1, x_2, \ldots, x_n \mid COND(x_1, x_2, \ldots, x_n, x_{n+1}, \ldots, x_{n+m})\}$

$x_1, x_2, \ldots, x_n, x_{n+1}, \ldots, x_{n+m}$ are domain variables that range over domains (of attributes) and COND is a condition or formula of the domain relational calculus.

A formula is made up of atoms and can be one of the following

1. $R(x_1, x_2, x_3, \ldots, x_j)$ where R is the name of relation and variables $x_1, x_2, \ldots x_j$ ranges over domains of corresponding attributes.
2. x_i OP x_j (OP comparison operator)
3. x_i OP c or c OP x_j

Eg Student

sid	sname	DOB	Dept	marks
a	..	.	d	e

$\{a, e \mid \exists(d)(Student(a, b, c, d, e) \wedge d = \text{'CS'})\}$

Q. Find the sid & marks of students of CS Dept.

Q. Find the sid of Students with there name and their department name.

$\{a\, b\, m \mid \exists(d)\, (Student(a\, b\, c\, d\, e) \wedge \exists(l)\, Department(l\, m\, n) \wedge d = l)\}$

Q Find the names of suppliers.

$\{b \mid Student(a, b, c)\}$

Q Find the suppliers whose rating is greater than 7.

$\{a\, b\, c \mid \exists(c)\, Student(a\, b\, c) \wedge c > 7\}$

Normalization

Informal Design guidelines for Relational Schemas

i) Semantics of the attributes
ii) Reducing the redundant values in tuples.
iii) Reducing the null values in the tuples.
iv) Disallowing the possibility of generating spurious tuples.

Guideline (i): Design a relation so that it is easy to explain its meaning. Do not combine attributes from multiple entity types into a single relation. Intuitively, if a relation schema corresponds to one entity type or one relationship type, the meaning tends to be clear. Otherwise, the relation corresponds to a mixture of multiple entities and relationships and hence becomes semantically unclear.

Guideline (ii): Design the base relation schemas so that no insertion, deletion, or modification anomalies are present in the relations. If any anomalies are present, note them clearly and make sure that the program that update the database will operate correctly.

One goal of schema design is to minimize storage space that the base relations (files) occupy.

Employee-Dept

(Ename, SSN, Addr, Dnumber, Dname, Dmgrssn)

If such table is stored as base table, then for every employee we will have department information and single 1 department can have many employee, so there will be repeated information of department in the table, hence, leading to wastage of storage space.

Another serious problem with using the above relation as base relation is the problem of update anomalies. These can be classified into: insertion anomalies, deletion anomalies, modification anomalies.

> Insertion anomalies

- To insert a new employee tuple, we must include either the attribute values for the department or nulls
- To insert new department tuple that has no employee as yet, we have only a way and that is to place null values in attributes of employee. This causes a problem because SSN is the primary key and each tuple is supposed to have employee entity.

> **Deletion Anomalies**

This problem is related to the second insertion anomaly situation discussed in page 32. If we delete from Employee-Dept an employee tuple that happens to represent the last employee working for a particular department, the information concerning that department is lost from the database.

> **Modification Anomalies**

In Employee-Dept, if we change the value of one of the attributes of a particular department - say, the manager of department 5 - we must update all the employee's tuples who work in that department; otherwise, the database will become inconsistent.

Guideline(iii) (Nulls are unexisting, unknown or not applied value). As far as possible, avoid placing attributes in a base relation whose values may frequently be null. If nulls are unavoidable, make sure that they apply in exceptional cases only and do not apply to a majority of tuples in the relation.

Guideline (iv): Design relation schema so that they can be joined with equality conditions on attributes that are either primary keys or foreign keys in a way that guarantees that no spurious tuples are generated.

- Do not have relations that contain matching attributes other than foreign key - primary key combinations.

Goal of Normalization.

a> 0% Redundancy.
b> Lossless join decomposition
c> Dependency preserving.

Functional Dependency

Let R be the relational schema, X, Y be the set of attributes of the relation R and t_1, t_2 are any two tuples of relation R. In relation R $X \rightarrow Y$ {X functionally determines Y or Y functionally determined by X} exists only if $t_1.X = t_2.X$ then $t_1.Y = t_2.Y$.

	X	Y
t_1	x_1	y_1
t_2	x_1	y_1

✓ $X \rightarrow Y$

X	Y
x_1	y
x_2	y

✓ $X \rightarrow Y$

X	Y
x_1	y_1
x_1	y_2

✗ $X \rightarrow Y$

Trivial Functional dependency

If $X \supseteq Y$, then $X \rightarrow Y$ is Trivial otherwise non trivial.

$AB \rightarrow A$
$AB \rightarrow B$
$AB \rightarrow AB$
} Trivial.

$A \rightarrow B$
$AB \rightarrow C$
$B \rightarrow AC$
} non-trivial.

Inference Rules for Functional dependencies

1. Reflexive Rule: if $X \supseteq Y$ then $X \to Y$. (trivial)
2. Augmentation rule: $\{X \to Y\} \models XZ \to YZ$.
3. Transitive rule: $\{X \to Y, Y \to Z\} \models X \to Z$.
4. Decomposition or projective rule: $\{X \to YZ\} \models X \to Y$ & $X \to Z$
5. Union or additive rule: $\{X \to Y, X \to Z\} \models X \to YZ$.
6. Pseudo transitive rule: $\{X \to Y, WY \to Z\} \models WX \to Z$.

Q. Identify all non trivial functional dependency of following relation

A	B	C
1	1	3
2	1	3
2	3	1
1	2	4
4	3	1
3	2	4
2	1	3

× $A \to B$ × $A \to C$ × $A \to BC$
✓ $B \to C$ × $B \to A$ × $B \to AC$
× $C \to A$ × $C \to B$ × $C \to AB$
✓ $AB \to C$
× $BC \to A$
✓ $AC \to B$

Attribute closure (X^+)

Set of attributes functionally determined by X.

eg R(ABCD) with $\{A \to B, B \to C, C \to D\}$

$A^+ = \{A, B, C, D\}$, $B^+ = \{B, C, D\}$

Q. $AB \to CD$, $AF \to D$, $DE \to F$, $C \to h$, $F \to E$, $h \to A$

which ~~is~~ are false.

i) $\{CF\}^+ = \{ACDEFh\}$
ii) $\{Bh\}^+ = \{ABCDh\}$
iii) $\{AF\}^+ = \{ACDEFh\}$
iv) $\{AB\}^+ = \{ACDFh\}$

Super key

Let R be the relational schema, X be the non-empty set of attributes over R.

If X^+ (closure set of X) determines all the attributes of relation R, then X is said to be Super key of R.

eg R(ABCD) $\{A \to B; B \to C, C \to D\}$

$A^+ = \{ABCD\}$, $\{AB\}^+ = \{ABCD$

A: Super key. ~~AB~~ Superkey.

i) A: CK, B: Not CK, AB Superkey not CK

ii) A: Not CK, B: CK, AB: SK Not CK,

iii) A: CK, B: CK, AB: Superkey Not CK

iv) A: Not CK, B: Not CK, AB: CK.

If any proper subset of X do not determine all the attributes of R then X can be called Candidate key.

eg R(ABCDE). $\{AB \to C, C \to D, B \to EA\}$

$\{AB\}^+ = \{ABCDE\}$ – NOT CK.

$\{A\}^+ = \{A\}$

$B^+ = \{BEACD\}$ – CK.

$C^+ = \{CD\}$ $E^+ = \{E\}$

Q. R (ABCDEF)
{BC->ADEF, A->BCDEF, B->F, D->E}
CK = {A, BC}.

Q. R (ABCD)
{A->B, B->C, C->D, D->A}
~~A+ = {A,B~~
CK : {A, B, C, D}

Q. R (ABCDEF)
{AB->E, C->D, D->E, E->F, F->A}
CK = {AB, FB, EB, DB, CB}.

Q. R (ABCDEF)
{AB->C, C->D, D->A}
CK: {ABE, DBE, CBE}

Functional Dependency Set Closure (F^+)

set of Functional dependencies logically implied in given FD set.

F^+ is logically equivalent to F.

F = {A->B, B->C}.

F^+ = { A->A, AC->B, A->B, AC->C, A->C, AB->C }.

Let F be the functional dependency and X->Y is logically implied in FD set F only if X^+ determines Y.

F = {A->B, B->C}. ⊨ A->C ✓
A^+ = {ABC}

F = {A->B, B->C} ⊨ B->A ✗
B^+ = {B, C}

Normalization

Normalization of data is a process of analyzing the given relation schemas based on their FDs and primary keys to achive the desirable properties of

1) minimizing redundancy and
2) minimizing the updation anomalies.

Unsatisfactory relation schemas that donot meet the normal form tests are dicompused into smaller relation schemas that meet the test.

Denormalization

The process of storing the join of higher normal form relations as a base relation - which is in a lower normal form - is known as denormalization.

The normal form of a relation refers to the highest normal form condition that it meets

First Normal form

Relation should have only atomic (single valued) attributes.

Problem: high redundancy level

Unnormalized data.

Sid	sname	cname	fee
S_1	A	C C++	5K 10K
S_2	B	C++ Java	10K 12K
S_3	B	Java	12K

↓ First Normal form.

sid	sname
S_1	A
S2	B
S3	B

sid - CK

sid	cname	fee
S_1	C	5K
S_1	C++	10K
S_2	C++	10K
S2	Java	12K
S3	Java	12K

(sid, cname) CK

Not in 2nd Normal form.

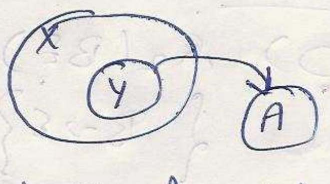
Second Normal Form

Relation R is in 2nd normal form only if.

i) R is in 1 NF
ii) R is not containing any non trivial Partial functional dependency.
(only full functional dependency)

Partial dependency

Let R be the Relation with attribute sets X, Y and A.



X : any CK
Y : Proper subset of any CK.
A : Non-Prime attribute

If $Y \rightarrow A$ exists in relation R. then, It is called Partial dependency.

eg. →CK

FD:{sid, cname → Fee,
cname → Fee}

Partial dependency.

It shows there is redundancy present between those attributes.

Sid	cname	Fee
S1	C	5K
S1	C++	10K
S2	C++	10K
S2	Java	12K
S3	Java	12K

⇓

Sid	cname
S1	C
S1	C++
S2	C++
S2	Java
S3	Java

cname	Fee
C	5K
C++	10K
Java	12K

eg R(ABCDE)

{AB→C, C→D, B→E}

AB - CK

B→E - P.D.

Not in 2NF

R_1(ABCD) R_2(BE)

{AB→C, C→D} {B→E} ✓2NF

AB: CK B: CK.

Third Normal Form

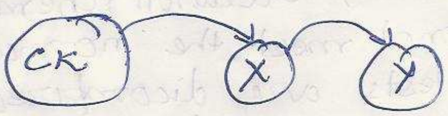
A Relation R is said to be in 3NF if for all the FDs X→Y either

i) X is a Super key

OR

ii) Y is a prime attribute.

Relation should not contain any Transitive dependency.



Transitive dependency.

R(ABCD) R_2(BE)

{AB→C, C→D}

⇓

ABC CD

AB→C C→D

Q. R(ABCDE,F)

{AB→BCDEF, BC→ADEF, D→E, B→F}

Sid	Sname	Pname	Pcost
S1	HCL	SMPS/MB	500/3500
S2	HP	MB/Processor	3500/7000
S3	Wipro	SMPS	500
S4	Inden	MB/Processor	3500/7000

43

Multivalued Dependency

Multivalued dependencies are the conceqvences of 1st NF, which disallowed an attribute in a tuple to have a set of value.

According to Navathe, a multi valued dependency (MVD) $X \twoheadrightarrow Y$ specified on relation schema R, where X and Y are both subsets of R, specifies the following construint on any relation state r of R: If two tuples t_1 and t_2 exist in r such that $t_1[X] = t_2[X]$, then two tuples t_3 and t_4 should also exist in r with the following properties, where we use Z to denote $(R-(X \cup Y))$:

-> $t_3[X] = t_4[X] = t_1[X] = t_2[X]$.

-> $t_3[Y] = t_1[Y]$ and $t_4[Y] = t_2[Y]$.

-> $t_3[Z] = t_2[Z]$ and $t_4[Z] = t_1[Z]$.

Eg

ENAME	PNAME	DNAME
RAM	A	CS
RAM	B	EC
RAM	A	EC
RAM	B	CS

ENAME $\twoheadrightarrow$ PNAM and ENAME $\twoheadrightarrow$ DNAME

ENAME	PNAME
RAM	A
RAM	B

ENAME	DNAME
RAM	CS
RAM	EC

Fourth Normal Form

A relation schema R is in 4NF with respect to a set of dependencies F if, for every nontrivial multivalued dependency $X \twoheadrightarrow Y$ in F^+, X is a Superkey of for R.

$X \twoheadrightarrow Y$ is called a trivial MVD if a) Y is subset of X or b) $X \cup Y = R$.

Join Dependency

A join dependency (JD), denoted by $JD(R_1, R_2, --, R_n)$ specified on relation schema R, specifies a constraint that every legal state of R should have a lossless join decomposition into $R_1, R_2, --, R_n$.

Notice that an MVD is a special case of JD where n=2.

Fifth NF

A relation schema R is is 5th NF or Project-join normal form (PJNF) with respect of a set F of functional, multivalued, and join dependencies if for every nontrivial join dependency JD in F^+, every R_i is SK in R.

(44)

JD (R1, R2, R3).

ENAME	PNAME	DNAME
RAM	X	D1
RAM	Y	D2
Situ	X	D2
Ravan	Y	D3
Situ	Z	D3
Situ	Y	D2
RAM	X	D2

R1

ENAME	PNAME
RAM	X
RAM	Y
Situ	X
Situ	Z
Ravan	Y

R2

ENAME	DNAME
RAM	D1
RAM	D2
Situ	D2
Situ	D3
RAVAN	D3

R3

PNAME	DNAME
X	D1
X	D2
Y	D2
Y	D3
Z	D3

R1, R2, R3 are in 5NF.

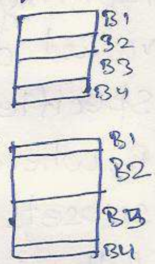
Physical database design

Indexing

Indexes are additional auxiliary access structure which are used to speedup the retrieval request.

- DB Files are devided into blocks.
- DB blocks devided into set of records.

DB File → Fixed length blocks / → Variable length blocks. (it has a DB File header)

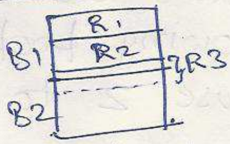


Records can also be fixed length and variable length.

Distribution of records in block.

Spanned organization.

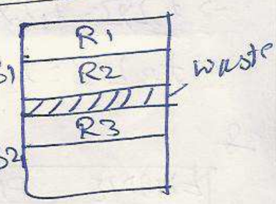
Block size - 100 B
Record size - 40 B



R3 spanned between two blocks.

Unspanned organization

A record may not be spanned between two blocks.



$$\text{Block Factor} = \left\lfloor \frac{\text{Block size}}{\text{Record size}} \right\rfloor = \left\lfloor \frac{100}{40} \right\rfloor = 2.$$

~~I/O~~ IO Cost: Number of blocks required to transfer from SM to MM to access record.

(45)

Indexing

- idea: Minimizing IO cost.
- size of index block is equal to size of DB File block.

Two entries only
<search key, Pointers>

Size of index file entry << size of record.

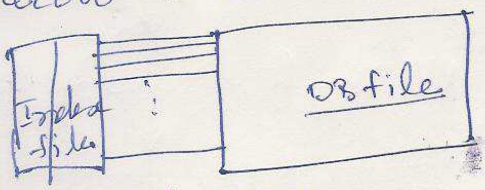
Block factor = no. of entries / block.

Index file block factor >> DB file blockfactor.

no of index block << # of DB file block.

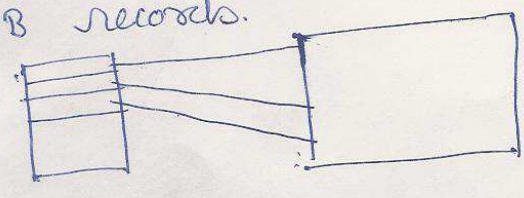
Categories ~~type~~ of indexing

1. Dense index
   → There should be an entry in the index file for every record in DB file.



one to one.

2. Sparse Index
   There is a single entry in Index file for set of DB records.



No. of Index file entries = No. of DB blocks.

Type of Indexing

1. Primary Index
   i) Search key used in index should also be used to Physically order the DB.
   ii) Search key should be PK or A.K.
   - can be < Dense / sparse.

   Atmost 1 primary index possible

2. Clustering Index
   i) Search key used in the index file should be used to physically order the DB.
   ii) Search key should be Nonkey.
   - always sparse index
   - atmost 1 clustering index possible.

3. Secondary Index
   i) Search key used in index file is not used to order the DB.
   ii) Search key may be PK, alternate key or Nonkey.
   - always dense index
   - can be built on key or non key
   - More than one secondary index possible.
   - Secondary index file size > primary index file size.

Query Processing

Query in High Level Language
↓
Scaning Parsing and Validating
↓
Intermediate form of query
↓
Query Optimizer
↓
Execution plan
↓
Query code generator
↓
code to execute the query
↓
Runtime Database Processor
↓
Result of query

Above are the typical steps when processing a high level query.

The scanner identifies the language tokens - such as SQL keywords, attribute names, and relation names - in the text of the query.

The parser checks the query syntax to determine whether it is formulated according to the syntax rules of the query language.

Validation is the process of checking that all attributes and relation names are valide and symentically meaningful names in the schema of the particular database being queried.

Intermediate form of query is in form of query tree or query graph.

Query optimizer chooses a suitable execution strategy for processing a query.

(59)

The query optimizer produces an execution plan and the code generator generates the code to execute that plan. The runtime database processor has the task of running the query code, whether in compiled or interpreted mode, to produce the query result.

If a runtime error results, an error message is generated by the runtime database processor.

Integrity Rules

If data losses integrity, it becomes garbage.

Domain Integrity

value of the collumn shuld be derived from the domain.

Entity Integrity

All values in Primary key must be not null and unique.

Refferential Integrity

A foreign key must be either null or must have a value that is derived from corresponding parrent key.

~~Sec~~

Security

- leagal and ethical issues
- policy issues at governmental institutional, corporate level.
- System related issues.
- Need in some organization to identify multiple security levels.

A DBMS typically includes a database security and authorization subsystem that is responsible for ensuring the security of the portions of the database against unauthorized access.

Two types of Database security machanism

- Discretionary Security mechanisms:

These are used to grant privile to users, including the capability to access specific datafiles, records or fields in a specific mode. (such as read, insert, delete & update).

(52)

Mandatory security mechanisms:

These are used to inforse multilevel security by classifying the data users into various security levels (or classes), and then implementing appropriate security policy of the organization.

The security mechanism of a DBMS must include provisions for restricting access to the database system as a whole. This function is called access control and is handled by creating user accounts and passwords to control the login process by the DBMS.

DB Security and the DBA

1 Account creation.
 - CREATE USER
2. Privilege granting:
 - GRANT
3. Privilege revocation
 - revoke
4. Security level assignment
 - GRANT

Recovery

Recovery from transaction failures usually means that the database is restored to the most recent consistant state just before the time of failure.

i) catastrophic failure.
 - copy of DB.

ii) Not physically damaged.
 - Undo

Shadow Paging

Shadow is used.
↓
A copy of original block is stored in the hard disc before it is sent to Main memory.

Assertion

CREATE ASSERTION Ass_sal CHECK (NOT EXISTS (Select * from EMP where EMP.Sal > 5000));

- a more general constraint which specify a condition which must be satisfied by every state of the Database.