Department of Commerce (CA)

CORE PAPER-II-DATABASE SYSTEM CONCEPTS

SEMESTER:I SUB CODE: 18MCC12C

M.COM(CA)

UNIT3: Embedded SQL-introduction-operations not involving cursors-involving cursors-Dyanamic statements-Query by example-Retreival operations-built-in-functions-update operations-QBE dictionary-normalization-functional dependency-first,second,third normal forms-relations with more than one candidate key-good and bad decompositions.

REFERENCE BOOK:

An introduction to database system-C.J. Dates

An introduction to database system-Bipin

PREPARED BY: DR. E.N. KANJANA,

ASST PROFESSOR.

Data Definition Language (DDL)

- Specification notation for defining the database schema

  Example: **create table** *instructor* (
  ID **char**(5),
  name **varchar**(20),
  dept_name **varchar**(20),
  salary **numeric**(8,2))

- DDL compiler generates a set of table templates stored in a ***data dictionary***
- Data dictionary contains metadata (i.e., data about data)
  - Database schema
  - Integrity constraints
    - Primary key (ID uniquely identifies instructors)
  - Authorization
    - Who can access what

Data Manipulation Language (DML)

- Language for accessing and manipulating the data organized by the appropriate data model
  - DML also known as query language
- Two classes of languages
  - **Pure** – used for proving properties about computational power and for optimization
    - Relational Algebra
    - Tuple relational calculus
    - Domain relational calculus
  - **Commercial** – used in commercial systems
    - SQL is the most widely used commercial language

SQL

v

- The most widely used commercial language
- SQL is NOT a Turing machine equivalent language
- SQL is NOT a Turing machine equivalent language
- To be able to compute complex functions SQL is usually embedded in some higher-level language
- Application programs generally access databases through one of
  - Language extensions to allow embedded SQL
  - Application program interface (e.g., ODBC/JDBC) which allow SQL queries to be sent to a database

Database Design

The process of designing the general structure of the database:

- Logical Design – Deciding on the database schema. Database design requires that we find a "good" collection of relation schemas.
  - Business decision – What attributes should we record in the database?
  - Computer Science decision – What relation schemas should we have and how should the attributes be distributed among the various relation schemas?
- Physical Design – Deciding on the physical layout of the database

Database Design (Cont.)

- Is there any problem with this relation?

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

Database Design (Cont.)

- Is there any problem with this relation?

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

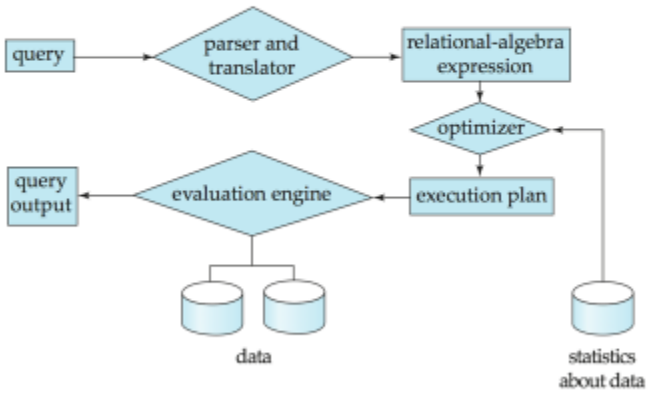
Database System Concepts - 6th Edition 1.29 ©Silberschatz, Korth and Sudarshan

Object-Relational Data Models

- Relational model: flat, "atomic" values
- Object Relational Data Models
  - Extend the relational data model by including object orientation and constructs to deal with added data types.
  - Allow attributes of tuples to have complex types, including non-atomic values such as nested relations.
  - Preserve relational foundations, in particular the declarative access to data, while extending modeling power.
  - Provide upward compatibility with existing relational languages.

Query Processing

1. Parsing and translation
2. Optimization
3. Evaluation



Introduction to Database Languages

The main objective of a database management system is to allow its users to perform a number of operations on the database such as insert, delete, and retrieve data in abstract terms without knowing
about the physical representations of data. To provide the various facilities to different types of users, a

DBMS normally provides one or more specialized programming languages called **Database** (or **DBMS**) **Languages**.

There are many popular RDBMS available to work. They are as follows:-

- MySQL
- MS SQL Server
- ORACLE
- MS ACCESS

SQL:-

SQL (Structured Query Language) is a database sublanguage for querying and modifying relational databases. It was developed by IBM Research in the mid 70's and standardized by ANSI in 1986.

SQL (pronounced "ess-que-el") stands for Structured Query Language. SQL is used to communicate with a database.

SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database. Some common relational database management systems that use SQL are: Oracle, Sybase, Microsoft SQL Server, Access, Ingres, etc.

Characteristics of SQL:-

- Allows users to describe the data.
- Allows users to define the data in database and manipulate that data.
- Allows embedding within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures, and views

SQL Functions:-

SQL has many built-in functions for performing calculations on data.

SQL Aggregate Functions

SQL aggregate functions return a single value, calculated from values in a column.

The Useful aggregate functions are as follows:

- AVG() - Returns the average value
- COUNT() - Returns the number of rows
- FIRST() - Returns the first value
- LAST() - Returns the last value
- MAX() - Returns the largest value
- MIN() - Returns the smallest value
- SUM() - Returns the sum

SQL Scalar functions

SQL scalar functions return a single value, based on the input value.

The Useful scalar functions are as follows:

- UCASE() - Converts a field to upper case
- LCASE() - Converts a field to lower case
- MID() - Extract characters from a text field
- LEN() - Returns the length of a text field
- ROUND() - Rounds a numeric field to the number of decimals specified
- NOW() - Returns the current system date and time
- FORMAT() - Formats how a field is to be displayed

Components of SQL:-

SQL commands are instructions used to communicate with the database to perform specific task that
work with data. SQL commands can be used not only for searching the database but also to perform various other functions like, for example, you can create tables, add data to tables, or modify data, drop
the table, set permissions for users. SQL commands are grouped into four major categories depending
on their functionality:

- **Data Definition Language (DDL)** - These SQL commands are used for creating, modifying, and dropping the structure of database objects. The commands are CREATE, ALTER, DROP, RENAME, and TRUNCATE.
- **Data Manipulation Language (DML)** - These SQL commands are used for storing, retrieving, modifying, and deleting data. These commands are SELECT, INSERT, UPDATE, and DELETE.
- **Transaction Control Language (TCL)** - These SQL commands are used for managing changes affecting the data. These commands are COMMIT, ROLLBACK, and SAVEPOINT.
- **Data Control Language (DCL)** - These SQL commands are used for providing security to database objects. These commands are GRANT and REVOKE.

Some of the Most Important SQL Commands

- **SELECT** - extracts data from a database
- **UPDATE** - updates data in a database
- **DELETE** - deletes data from a database
- **INSERT INTO** - inserts new data into a database
- **CREATE DATABASE** - creates a new database
- **ALTER DATABASE** - modifies a database

CREATE TABLE - creates a new table

- **ALTER TABLE** - modifies a table
- **DROP TABLE** - deletes a table
- **CREATE INDEX** - creates an index (search key)
- **DROP INDEX** - deletes an index

Some of the Most Important SQL Commands with SQL statement

DML: Data Manipulation Language

SQL-Data Statements -- query and modify tables and columns

- SELECT Statement -- query tables and views in the database
- INSERT Statement -- add rows to tables
- UPDATE Statement -- modify columns in table rows
- DELETE Statement -- remove rows from tables

TCL:- Transaction Control Language

SQL-Transaction Statements -- control transactions

- COMMIT Statement -- commit the current transaction
- ROLLBACK Statement -- roll back the current transaction

DDL:- Data Definition Language

SQL-Schema Statements -- maintain schema (catalog)

- CREATE TABLE Statement -- create tables
- CREATE VIEW Statement -- create views

- DROP TABLE Statement -- drop tables
- DROP VIEW Statement -- drop views
- GRANT Statement -- grant privileges on tables and views to other users
- REVOKE Statement -- revoke privileges on tables and views from other users

The SQL SELECT Statement:-

The SELECT statement is used to select data from a database.

Syntax:

SELECT *column_name,column_name*
FROM *table_name*;
Or
SELECT * FROM *table_name*;

WHERE clause: - It is used to filter records.

SELECT *column_name,column_name*
FROM *table_name*
WHERE *column_name operator value*;

INSERT INTO *table_name* (*column1*,*column2*,*column3*,...)
VALUES (*value1*,*value2*,*value3*,...);

SQL UPDATE Statement

The UPDATE statement is used to update records in a table.

UPDATE *table_name*
SET *column1=value1*,*column2=value2*,...
WHERE *some_column=some_value*;

SQL DELETE Statement

The DELETE statement is used to delete records in a table.

DELETE FROM *table_name*
WHERE *some_column=some_value*;

SQL CREATE TABLE Statement

The CREATE TABLE statement is used to create a table in a database.
Tables are organized into rows and columns; and each table must have a name.

CREATE TABLE *table_name*
(
column_name1 data_type(*size*),
column_name2 data_type(*size*),
column_name3 data_type(*size*),
....
);

The DROP TABLE Statement

The DROP TABLE statement is used to delete a table.

DROP TABLE table_name;

The ALTER TABLE Statement

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.
To add a column in a table, use the following syntax:

ALTER TABLE table_name
ADD column_name datatype

SQL GRANT Command

SQL GRANT is a command used to provide access or privileges on the database objects to the users.

The Syntax for the GRANT command is:

GRANT privilege_name
ON object_name
TO {user_name |PUBLIC |role_name}
[WITH GRANT OPTION];

- *privilege_name* is the access right or privilege granted to the user. Some of the access rights are ALL, EXECUTE, and SELECT.
- *object_name* is the name of an database object like TABLE, VIEW, STORED PROC and SEQUENCE.
- *user_name* is the name of the user to whom an access right is being granted.
- *user_name* is the name of the user to whom an access right is being granted.
- *PUBLIC* is used to grant access rights to all users.
- *ROLES* are a set of privileges grouped together.

SQL Operator

An operator is a reserved word or a character used primarily in an SQL statement's WHERE clause to
perform operation(s), such as comparisons and arithmetic operations.

Operators are used to specify conditions in an SQL statement and to serve as conjunctions for multiple
conditions in a statement.

- Arithmetic operators
- Comparison operators
- Logical operators
- Operators used to negate conditions

Set operators:-

SQL support few of set operators on the SQL tables. They are as follows:-

- ✓ Union
- ✓ Intersect

ORDER BY :-

The SQL **ORDER BY** clause is used to sort the data in ascending or descending order, based on one or
more columns. Some database sorts query results in ascending order by default.

Syntax:-

SELECT column-list
FROM table_name
[WHERE condition]
[ORDER BY column1, column2, .. columnN] [ASC | DESC];

GROUP BY

The SQL **GROUP BY** clause is used in collaboration with the SELECT statement to arrange identical data
into groups.

The GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY
clause.

Syntax:

```
SELECT column1, column2
FROM table_name
WHERE [ conditions ]
GROUP BY column1, column2
ORDER BY column1, column2
```

HAVING

The HAVING clause enables you to specify conditions that filter which group results appear in the final
results.

The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.

Syntax

```
SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY
SELECT column1, column2
FROM table1, table2
WHERE [ conditions ]
GROUP BY column1, column2
HAVING [ conditions ]
ORDER BY column1, column2
```

Subquery

A Subquery or Inner query or Nested query is a query within another SQL query and embedded within
the WHERE clause.

A subquery is used to return data that will be used in the main query as a condition to further restrict
the data to be retrieved.

ubqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN etc.

There are a few rules that subqueries must follow:

- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
- An ORDER BY cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY can be used to perform the same function as the ORDER BY in a subquery.
- Subqueries that return more than one row can only be used with multiple value operators, such as the IN operator.
- The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.
- A subquery cannot be immediately enclosed in a set function.

• The BETWEEN operator cannot be used with a subquery; however, the BETWEEN operator can be used within the subquery.

```
SELECT column_name [, column_name ]
FROM table1 [, table2 ]
WHERE column_name OPERATOR
(SELECT column_name [, column_name ]
FROM table1 [, table2 ]
[WHERE])
```

Join

The SQL **Joins** clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

SQL Join Types:

There are different types of joins available in SQL:

• INNER JOIN: returns rows when there is a match in both tables.
• LEFT JOIN: returns all rows from the left table, even if there are no matches in the right table.
• RIGHT JOIN: returns all rows from the right table, even if there are no matches in the left table.
• FULL JOIN: returns rows when there is a match in one of the tables.
• SELF JOIN: is used to join a table to itself as if the table were two tables, temporarily renaming at least
one table in the SQL statement.
• CARTESIAN JOIN: returns the Cartesian product of the sets of records from the two or more joined tables.

SQL Functions:-

There are two types of functions in SQL

1) Single Row Functions: Single row or Scalar functions return a value for every row that is processed in
a query.

2) Group Functions: These functions group the rows of data based on the values returned by the query.

This is discussed in SQL GROUP Functions. The group functions are used to calculate aggregate values
like total or average, which return just one total or one average value after processing a group of rows.

There are four types of single row functions.

Functional dependency

In a given table, an attribute *Y* is said to have a functional dependency on a set of attributes *X* (written $X \rightarrow Y$) if and only if each *X* value is associated with precisely one *Y* value.

For example, in an "Employee" table that includes the attributes "Employee ID" and "Employee Date of
Birth", the functional dependency {Employee ID} → {Employee Date of Birth} would hold. It follows from the previous two sentences that each {Employee ID} is associated with precisely one {Employee
Date of Birth}.

Full functional dependency

An attribute is fully functionally dependent on a set of attributes X if it is:
• functionally dependent on X, and
• not functionally dependent on any proper subset of X. {Employee Address} has a functional dependency on {Employee ID, Skill}, but not a *full* functional dependency, because it is also dependent on {Employee ID}.Even by the removal of {Skill} functional dependency still holds between {Employee Address} and {Employee ID}.

Transitive dependency

A transitive dependency is an indirect functional dependency, one in which $X \rightarrow Z$ only by virtue of $X \rightarrow Y$ and $Y \rightarrow Z$.

Trivial functional dependency

A trivial functional dependency is a functional dependency of an attribute on a superset of itself. {Employee ID, Employee Address} → {Employee Address} is trivial, as is {Employee Address} → {Employee Address}.

Multivalve dependency

A multivalued dependency is a constraint according to which the presence of certain rows in a table implies the presence of certain other rows.

Join dependency

A table *T* is subject to a join dependency if *T* can always be recreated by joining multiple tables each having a subset of the attributes of *T*.

Normalization Process

• Relations can fall into one or more categories (or classes) called Normal Forms
• Normal Form: A class of relations free from a certain set of modification anomalies.
• Normal forms are given names such as:
- First normal form (1NF)
- Second normal form (2NF)
- Third normal form (3NF)
- Boyce-Codd normal form (BCNF)
- Fourth normal form (4NF)
- Fifth normal form (5NF)
- Domain-Key normal form (DK/NF)

• These forms are cumulative. A relation in Third normal form is also in 2NF and 1NF.
• The Normalization Process for a given relation consists of:
a. Specify the Key of the relation
b. Specify the functional dependencies of the relation.
Sample data (tuples) for the relation can assist with this step.
c. Apply the definition of each normal form (starting with 1NF).
d. If a relation fails to meet the definition of a normal form, change the relation (most often by splitting the relation into two new relations) until it meets the definition.
e. Re-test the modified/new relations to ensure they meet the definitions of each normal form.
In the next set of notes, each of the normal forms will be defined along with an example of the normalization steps.

First Normal Form (1NF)

A relation is in first normal form if it meets the definition of a relation:
- Each attribute (column) value must be a single value only.
- All values for a given attribute (column) must be of the same type.

- Each attribute (column) name must be unique.
- The order of attributes (columns) is insignificant
- No two tuples (rows) in a relation can be identical.
- The order of the tuples (rows) is insignificant.

• If you have a key defined for the relation, then you can meet the unique row requirement.
• Example relation in 1NF (note that key attributes are underlined):
• STOCKS (Company, Symbol, Headquarters, Date, Close_Price)

Second Normal Form (2NF)

A relation is in second normal form (2NF) if all of its non-key attributes are dependent on all of the *key*.

Another way to say this: A relation is in second normal form if it is free from partial-key dependencies

Relations that have a single attribute for a key are automatically in 2NF.

This is one reason why we often use artificial identifiers (non-composite keys) as keys.

In the example below, Close Price is dependent on Company, Date

The following example relation *is not* in 2NF:

STOCKS (Company, Symbol, Headquarters, Date, Close_Price)

Third Normal Form (3NF)

A relation is in third normal form (3NF) if it is in second normal form and it contains no *transitive dependencies*.

Consider relation R containing attributes A, B and C. R(A, B, C)

If A → B and B → C then A → C

Transitive Dependency: Three attributes with the above dependencies.

Example: At CUNY:

Course_Code → Course_Number, Section

Course_Number, Section → Classroom, Professor

Boyce-Codd Normal Form (BCNF)

• A relation is in BCNF if every determinant is a candidate key.
• Recall that not all determinants are keys.
• Those determinants that are keys we initially call *candidate keys*.
• Eventually, we select a single candidate key to be *the key* for the relation

Fourth Normal Form (4NF)

A relation is in fourth normal form if it is in BCNF and it contains no *multivalued dependencies*.

Multivalued Dependency: A type of functional dependency where the determinant can determine more than one value.

More formally, there are 3 criteria:

1. There must be at least 3 attributes in the relation. call them A, B, and C, for example.
2. Given A, one can determine multiple values of B.
3. Given A, one can determine multiple values of C.
4. B and C are independent of one another.

Book example:

Student has one or more majors.
Student participates in one or more activities.

StudentID	Major	Activities
100	CIS	Baseball
100	CIS	Volleyball
100	Accounting	Baseball
100	Accounting	Volleyball
200	Marketing	Swimming

FD1: StudentID →→ Major
FD2: StudentID →→ Activities

Portfolio ID	Stock Fund	Bond Fund
999	Janus Fund	Municipal Bonds
999	Janus Fund	Dreyfus Short-Intermediate Municipal Bond Fund

999	Scudder Global Fund	Municipal Bonds
999	Scudder Global Fund	Dreyfus Short-Intermediate Municipal Bond Fund
888	Kaufmann Fund	T. Rowe Price Emerging Markets Bond Fund

A few characteristics:

- No regular functional dependencies
- All three attributes taken together form the key.
- Latter two attributes are independent of one another.
- Insertion anomaly: Cannot add a stock fund without adding a bond fund (NULL Value). Must always maintain the combinations to preserve the meaning.

• Stock Fund and Bond Fund form a multivalued dependency on Portfolio ID.
• PortfolioID →→ Stock Fund
• PortfolioID →→ Bond Fund

Resolution: Split into two tables with the common key:

Portfolio ID	Stock Fund
999	Janus Fund
999	Scudder Global Fund
888	Kaufmann Fund

Portfolio ID	Bond Fund
999	Municipal Bonds
999	Dreyfus Short-Intermediate Municipal Bond Fund
888	T. Rowe Price Emerging Markets Bond Fund

Fifth Normal Form (5NF)
Also called “Projection Join” Normal form.
There are certain conditions under which after decomposing a relation, it cannot be reassembled back
into its original form.
We don’t consider these issues here.
Domain Key Normal Form (DK/NF)
A relation is in DK/NF if every *constraint* on the relation is a logical consequence of the definition of *keys*and *domains*.
Constraint: An rule governing static values of an attribute such that we can determine if this constraint
is True or False. Examples:

- Functional Dependencies
- Multivalued Dependencies
- Inter-relation rules

Key: Unique identifier of a tuple.
Domain: The physical (data type, size, NULL values) and semantic (logical) description of what values
an attribute can hold.
There is no known algorithm for converting a relation directly into DK/NF.
What is Normalization?
Normalization is the process of efficiently organizing data in a database. There are two goals of the normalization process: eliminating redundant data (for example, storing the same data in more than
one table) and ensuring data dependencies make sense (only storing related data in a table). Both of
these are worthy goals as they reduce the amount of space a database consumes and ensure that data is
logically stored.
Summary of the Normal Forms
The database community has developed a series of guidelines for ensuring that databases are normalized. These are referred to as normal forms and are numbered from one (the lowest form of normalization, referred to as first normal form or 1NF) through five (fifth normal form or 5NF). In practical applications, you'll often see 1NF, 2NF, and3NF along with the occasional 4NF. Fifth normal
form is very rarely seen and won't be discussed in this article.
Before we begin our discussion of the normal forms, it's important to point out that they are guidelines
and guidelines only. Occasionally, it becomes necessary to stray from them to meet practical business
requirements. However, when variations take place, it's extremely important to evaluate any possible
ramifications they could have on your system and account for possible inconsistencies. That said, let's

explore the normal forms.

First Normal Form (1NF)

First normal form (1NF) sets the very basic rules for an organized database:
Eliminate duplicative columns from the same table.
Create separate tables for each group of related data and identify each row with a unique column or set of columns (the primary key).

Second Normal Form (2NF)

Second normal form (2NF) further addresses the concept of removing duplicative data:
Meet all the requirements of the first normal form.
Remove subsets of data that apply to multiple rows of a table and place them in separate tables.
Create relationships between these new tables and their predecessors through the use of foreign keys.

Third Normal Form (3NF)

Third normal form (3NF) goes one large step further:
Meet all the requirements of the second normal form.
Remove columns that are not dependent upon the primary key

Boyce-Codd Normal Form (BCNF or 3.5NF)

The Boyce-Codd Normal Form, also referred to as the "third and half (3.5) normal form", adds one more
requirement:
Meet all the requirements of the third normal form.
Every determinant must be a candidate key.

Fourth Normal Form (4NF)

Finally, fourth normal form (4NF) has one additional requirement:
Meet all the requirements of the third normal form.
A relation is in 4NF if it has no multi-valued dependencies.
Remember, these normalization guidelines are cumulative. For a database to be in 2NF, it must first fulfill all the criteria of a 1NF database.