Lecture 5 Notes

Normalization + SQL Group By Clause

Discussion Points:

- Normalization
- Definitions of Keys and Attributes Participating in Keys
- First Normal Form
- Second Normal Form
- Third Normal Form
- BCNF
- Fourth Normal Form
- Fifth Normal Form
- SQL: Group By

❖ Normalization

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

1. Lossless-join Decomposition or nonadditive join (complete reproduction)
2. No Redundancy (BCNF or 3NF)
3. Dependency Preservation
4. Minimizing the insertion, deletion, and update anomalies.

We assume that a set of functional dependencies is given for each relation, and that each relation has a designated primary key.
More general definitions of these normal forms, which take into account all candidate keys of a relation rather than just the primary key.

Normalization of Relations

The normalization process, as first proposed by Codd (1972a),takes a relation schema through a series of tests to *certify* whether it satisfies a certain **normal form**.

Initially, Codd proposed three normal forms, which he called first,second, and third normal form.

A stronger definition of 3NF—called Boyce-Codd normal form(BCNF)—was proposed later by Boyce and Codd. All these normal forms are based on a single analytical tool: the functional dependencies among the attributes of a relation.

Definition.

The **normal form** of a relation refers to the highest normal form condition that it meets, and hence indicates the degree to which it has been normalized.
Normal forms, when considered *in isolation* from other factors, donot guarantee a good database design. It is generally not sufficient to check separately that each relation schema in the database is in a given normal form.
Rather, the process of normalization through decomposition mustalso confirm the existence of additional properties that the relational schemas, taken together, should possess. These wouldinclude two properties:

Practical Use of Normal Forms

Most practical design projects acquire existing designs of databases from previous designs, designs in legacy models, orfrom existing files. Normalization is carried out in practice so that the resulting designs are of high quality and meet the desirable propertiesstated previously.
Although several higher normal forms have been defined, database design as practiced in industry today pays particular attention to normalization only up to 3NF, BCNF, or at most 4NF.

Another point worth noting is that the database designers *need not* normalize to the highest possible normal form. Relations maybe left in a lower normalization status, such as 2NF, for performance reason.

Denormalization is the process of storing the join of highernormal form relations as a base relation, which is in a lowernormal form.

❖ Definitions of Keys and Attributes Participating in Keys

Definition: A **superkey** of a relation schema $R = \{A1, A2, \ldots, An\}$is a set of attributes S -> R with the property that no two tuples $t1$ and $t2$ in any legal relation state r of R will have $t1[S] = t2[S]$.
A **key** K is a superkey with the additional property that removal ofany attribute from K will cause K not to be a superkey anymore.
The difference between a key and a superkey is that a key has tobe *minimal;* that is, if we have a key $K = \{A1, A2, ..., Ak\}$ of R, then $K - \{Ai\}$ is not a key of R for any

Ai, $1 \leq i \leq k$

{Ssn} is a key for EMPLOYEE, whereas {Ssn}, {Ssn, Ename}, {Ssn, Ename, Bdate}, and any set of attributes that includes Ssn are all superkeys.

If a relation schema has more than one key, each is called a candidate key.

One of the candidate keys is *arbitrarily* designated to be the **primary key**, and the others are called secondary keys.

Definition:

An attribute of relation schema R is called a **prime attribute** of R if it is a member of *some candidate key* of R.

An attribute is called **nonprime** if it is not a prime attribute—thatis, if it is not a member of any candidate key, both Ssn and Pnumber are prime attributes of WORKS_ON, whereas other attributes of WORKS_ON are nonprime.

❖ First Normal Form

First normal form (1NF) is now considered to be part of the formal definition of a relation in the basic (flat) relational model.It states that:

1. the domain of an attribute must include only *atomic* (simple,indivisible) *values* and
2. that the value of any attribute in a tuple must be a *singlevalue* from the domain of that attribute.

Hence, 1NF disallows having a set of values, a tuple of values, ora combination of both as an attribute value for a *single tuple.* In other words, 1NF disallows *relations within relations* or *relations as attribute values within tuples*.
The only attribute values permitted by 1NF are single **atomic** (or indivisible) values.

Consider the DEPARTMENT relation schema, whose primary keyis Dnumber, and suppose that we extend it by including the Dlocations attribute. We assume that each department can have a number of locations.

As we can see, this is not in 1NF because Dlocations is not anatomic attribute. There are two ways we look at the Dlocations attribute:

The domain of Dlocations contains atomic values, but some tuples can have a set of these values. In this case, Dlocations is not functionally dependent on the primary key Dnum.

First normal form also disallows multi-valued attributes that are themselves composite. These are called **nested relations** because each tuple can have a relation *within it.*
This procedure can be applied recursively to a relation with multiple-level nesting to **unnest** the relation into a set of 1NF relations. This is useful in converting an unnormalized relationschema with many levels of nesting into 1NF relations.

❖ Second Normal Form

Second normal form (2NF) is based on the concept of *full functional dependency.*
Functional Dependency:

The attribute B is fully functionally dependent on the attribute A if each value of A determines one and only one value of B.

Example: PROJ_NUM - >PROJ_NAME
In this case, the attribute PROJ_NUM is known as the determinant attribute and the attribute PROJ_NAME is known asthe dependent attribute.

Generalized Definition:

Attribute A determines attribute B (that is B is functionally dependent on A) if all of the rows in the table that agree in valuefor attribute A also agree in value for attribute B.

Fully functional dependency (composite key)

If attribute B is functionally dependent on a composite key A butnot on any subset of that composite key, the attribute B is fully functionally dependent on A.

Partial Dependency:

When there is a functional dependence in which the determinant is only part of the primary key, then there is a partial dependency.

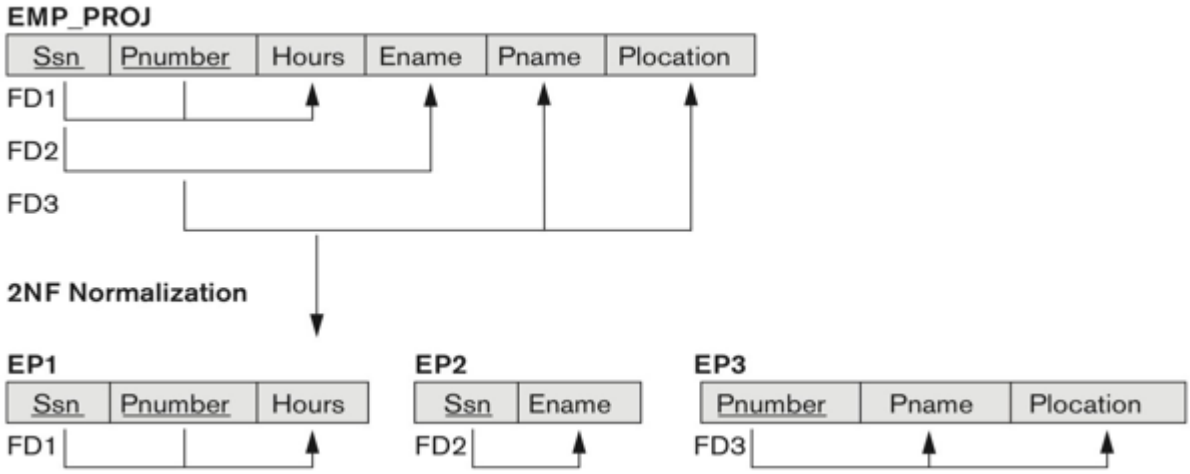
For example:

If (A, B) -> (C, D) and B -> C and (A, B) is the primary key, then the functional dependence B -> C is a partial dependency.
{Ssn, Pnumber} -> Hours is a full dependency

(neither Ssn -> Hours nor Pnumber –> Hours holds).

However, the dependency {Ssn, Pnumber} -> Ename is partial because Ssn -> Ename holds.

Example:



Transitive Dependency:
When there are the following functional dependencies such that X->Y, Y-> Z and X is the primary key, then X -> Z is a transitive dependency because X determines the value of Z via Y. Whenever a functional dependency is detected amongst non- prime, there is a transitive dependency.

Definition. A relation schema *R* is in 2NF if every nonprime attribute *A* in *R* is *fully functionally dependent* on the primary keyof *R*.
The test for 2NF involves testing for functional dependencieswhose left-hand side attributes are part of the primary key.
If the primary key contains a single attribute, the test need not beapplied at all.

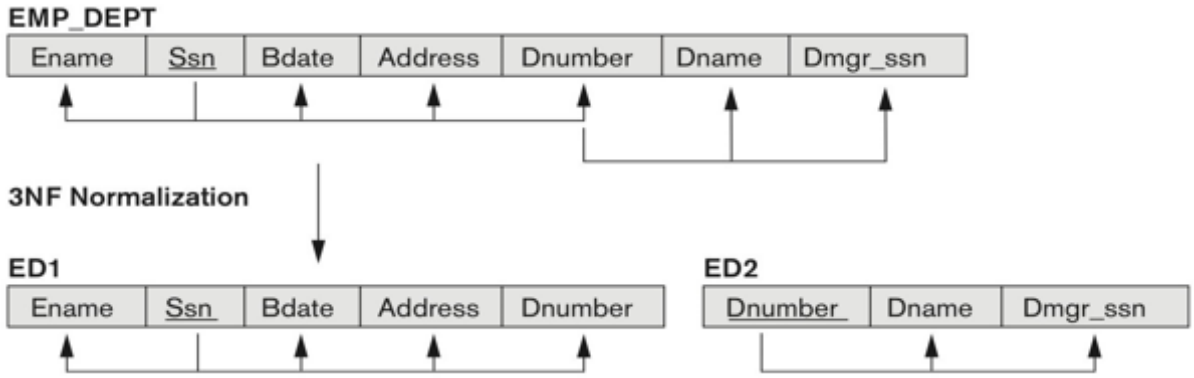
If a relation schema is not in 2NF, it can be *second normalized* or *2NF normalized* into a number of 2NF relations in which nonprimeattributes are associated only with the part of the primary key on which they are fully functionally dependent.

❖ Third Normal Form

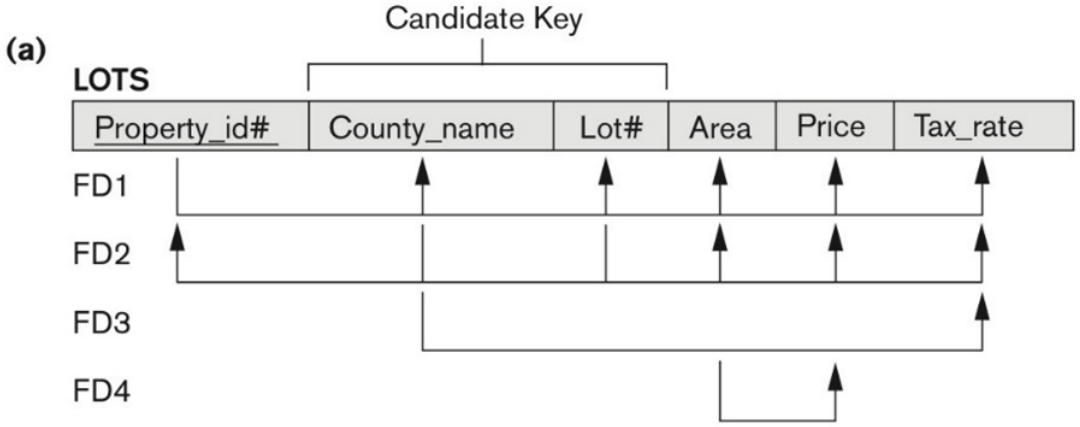
Third normal form (3NF) is based on the concept of *transitivedependency*.
A functional dependency *X ->Y* in a relation schema *R* is a transitive dependency if there exists a set of attributes Z in R that is neither a candidate key nor a subset of any key of R, and both X->Z and Z->Y hold.

Definition. According to Codd's original definition, a relation schema *R* is in **3NF** if it satisfies 2NF *and* no nonprime attribute of R is transitively dependent on the primary key.

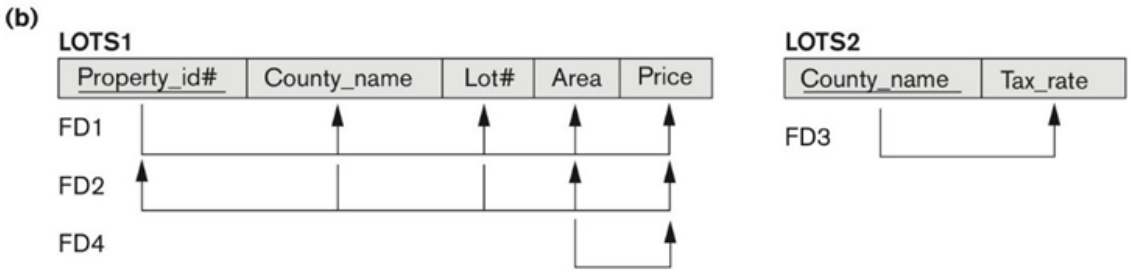
Example:



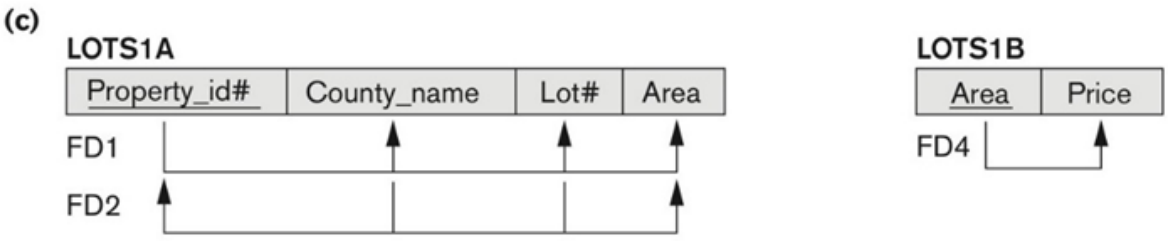
Example:



2NF



3NF



General Definitions of Second and Third Normal Forms

General Definition of Second Normal Form

A relation schema *R* is in **second normal form (2NF)** if every nonprime attribute *A* in *R* is not partially dependent on *any* key of R.

The test for 2NF involves testing for functional dependencies whose left-hand side attributes are *part of* the primary key. If theprimary key contains a single attribute, the test need not be applied at all.

General Definition of Third Normal Form

A relation schema *R* is in **third normal form (3NF)** if, whenever a *nontrivial* functional dependency *X*->*A* holds in *R*, either

1. *X* is a superkey of *R*, or
2. *A* is a prime attribute of *R*.

Interpreting the General Definition of Third Normal Form

A relation schema *R* violates the general definition of 3NF if a functional dependency *X* -> *A* holds in *R* that does not meet eithercondition—meaning that it violates *both* conditions (a) and (b) of 3NF. This can occur due to two types of problematic functional dependencies:
A nonprime attribute determines another nonprime attribute. Here we typically have a transitive dependency that violates 3NF.

A proper subset of a key of *R* functionally determines a nonprime attribute. Here we have a partial dependency that violates 3NF (and also 2NF).
Therefore, we can state a **general alternative definition of 3NF** as follows:
Alternative Definition. A relation schema *R* is in 3NF if everynonprime attribute of *R* meets both of the following conditions:

- It is fully functionally dependent on every key of *R*.
- It is non-transitively dependent on every key of *R*.

❖ Boyce-Codd Normal Form

Definition: A relation schema *R* is in **BCNF** if whenever a nontrivial functional dependency X -> A holds in R, then X is a super key of R. In practice, most relation schemas that are in 3NF are also in BCNF.

Only if *X*->*A* holds in a relation schema *R* with *X* not being a super key *and A* being a prime attribute will *R* be in 3NF but not in BCNF. Ideally, relational database design should strive to achieveBCNF or 3NF for every relation schema.

Summary of Normal Forms Based on Primary Keys andCorresponding Normalization

Normal Form	Test	Remedy (Normalization)
First (1NF)	Relation should have no multivalued attributes or nested Relations	Form new relations for each multivalued attribute or nested relation.
Second (2NF)	For relations where primary key contains multiple attributes, no non-key attribute should be functionally dependent on a part of the primary key.	Decompose and set up a new relation for each partial key with its dependent attribute(s).Make sure to keep a relation with the original primary key and any attributes that are fullyfunctionally dependent on it.
Third (3NF)	Relation should not have a non-key attribute functionally determined by another non-key attribute (or by a set of non-key attributes). That is, there should be no transitive dependency of a non-key attribute on the primary Key.	Decompose and setup a relation that includes the non-key attribute(s) that functionally determine(s) other non-key attribute(s).

❖ Fourth Normal Form

Definition. A multivalued dependency (MVD) X —>> Y specified on relation schema R, where X and Y are both subsets of R, specifies the following constraint on any relation state r of R: If two tuples t1 and t2 exist in r such that t1[X] = t2[X], then two tuples t3 and t4 should also exist in r with the following properties, where we use Z to denote (R 2 (X υ Y)):

- t3[X] = t4[X] = t1[X] = t2[X].
- t3[Y] = t1[Y] and t4[Y] = t2[Y].
- t3[Z] = t2[Z] and t4[Z] = t1[Z].

An MVD X —>> Y in R is called a trivial MVD if (a) Y is a subset of X, or (b) X υ Y = R.

Example:

(a) EMP

Ename	Pname	Dname
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

(b) EMP_PROJECTS

Ename	Pname
Smith	X
Smith	Y

EMP_DEPENDENTS

Ename	Dname
Smith	John
Smith	Anna

The EMP relation with two MVDs: Ename –>> Pname and Ename –>> Dname. So, Decomposing the EMP relation into two 4NF relations EMP_PROJECTS and EMP_DEPENDENTS.

❖ Fifth Normal Form

Definition. A join dependency (JD), denoted by JD(R1, R2, ..., Rn), specified on relation schema R, specifies a constraint on the states r of R.
The constraint states that every legal state r of R should have a non-additive join decomposition into R1, R2, ..., Rn; that is, for every such r we have

$$* (\pi_{R1}(r), \pi_{R2}(r), \ldots, \pi_{Rn}(r)) = r$$

Note: an MVD is a special case of a JD where n = 2.
A join dependency JD(R1, R2, ..., Rn), specified on relation schema R, is a trivial JD if one of the relation schemas Ri in JD(R1, R2, ..., Rn) is equal to R.

Example:

SUPPLY

Sname	Part_name	Proj_name
Smith	Bolt	ProjX
Smith	Nut	ProjY
Adamsky	Bolt	ProjY
Walton	Nut	ProjZ
Adamsky	Nail	ProjX
Adamsky	Bolt	ProjX
Smith	Bolt	ProjY

R_1

Sname	Part_name
Smith	Bolt
Smith	Nut
Adamsky	Bolt
Walton	Nut
Adamsky	Nail

R_2

Sname	Proj_name
Smith	ProjX
Smith	ProjY
Adamsky	ProjY
Walton	ProjZ
Adamsky	ProjX

R_3

Part_name	Proj_name
Bolt	ProjX
Nut	ProjY
Bolt	ProjY
Nut	ProjZ
Nail	ProjX

The relation SUPPLY with no MVDs is in 4NF but not in 5NF if it has the JD(R1, R2, R3). So, decomposing the relation SUPPLY into the 5NF relations R1, R2, R3.

❖ SQL Group By

- Group functions operate on sets of rows to give one result per group.
- *Types of Group Functions*
    - AVG
    - COUNT
    - MAX
    - MIN
    - STDDEV
    - SUM
    - VARIANCE

- *Group Functions Syntax*

  SELECT [column,] group_function(column), ...
  FROM table
  [WHERE condition]
  [GROUP BY column]
  [ORDER BY column];

- *Using the AVG and SUM Functions*

    - You can use AVG and SUM for numeric data.

      SELECT AVG(salary), MAX(salary),
      MIN(salary), SUM(salary)
      FROM employees
      WHERE job_id LIKE '%REP%';

- *Using the MIN and MAX Functions*

  SELECT MIN(hire_date), MAX(hire_date)
  FROM employees;

- *Using COUNT Function*
    - COUNT(*) returns the number of rows in a table

      SELECT COUNT(*)
      FROM employees
      WHERE department_id = 50;

    - COUNT(expr) returns the number of rows with non-null values for the expr.
    - Display the number of department values in the EMPLOYEES table, excluding null values.

SELECT COUNT(commission_pct)
FROM employees
WHERE department_id = 80;

- *Using the DISTINCT Keyword*
  - COUNT(DISTINCT expr) returns the number of distinct non-null values of the expr.
  - Display the number of distinct department values in the EMPLOYEES table.

    SELECT COUNT(DISTINCT department_id)
    FROM employees;

- *Group Functions and Null Values*
  - Group functions ignore null values in the column

    SELECT AVG(commission_pct)
    FROM employees;

- *Using the NVL Function with Group Functions*
  - The NVL function forces group functions to include null values

    SELECT AVG(NVL(commission_pct, 0))
    FROM employees;

- *Creating Groups of Data: The GROUP BY Clause Syntax*

  - Divide rows in a table into smaller groups by using the GROUP BY clause.

    SELECT column, group_function(column)
    FROM table
    [WHERE condition]
    [GROUP BY group_by_expression]
    [ORDER BY column];

- *Using the GROUP BY Clause*
  - All columns in the SELECT list that are not in group functions must be in the GROUP BY clause.

    SELECT department_id, AVG(salary)
    FROM employees
    GROUP BY department_id ;

- The GROUP BY column does not have to be in the SELECT list.

  SELECT AVG(salary)
  FROM employees
  GROUP BY department_id ;

- *Using the GROUP BY Clause on Multiple Columns*

  SELECT department_id dept_id, job_id, SUM(salary)
  FROM employees
  GROUP BY department_id, job_id ;

- *Excluding Group Results: The HAVING Clause*

  - You cannot use the WHERE clause to restrict groups.
  - Use the HAVING clause to restrict groups.
  - You cannot use group functions in the WHERE clause.
  - Use the HAVING clause to restrict groups:
    - 1. Rows are grouped.
    - 2. The group function is applied.
    - 3. Groups matching the HAVING clause are displayed.

  SELECT column, group_function
  FROM table
  [WHERE condition]
  [GROUP BY group_by_expression]
  [HAVING group_condition]
  [ORDER BY column];

  Example:

  SELECT department_id, MAX(salary)
  FROM employees
  GROUP BY department_id
  HAVING MAX(salary)>10000 ;

  Example:

  SELECT job_id, SUM(salary) PAYROLL
  FROM employees
  WHERE job_id NOT LIKE '%REP%'
  GROUP BY job_id
  HAVING SUM(salary) > 13000
  ORDER BY SUM(salary);

- *Nesting Group Functions*

    - Display the maximum average salary.

```
SELECT MAX(AVG(salary))
FROM employees
GROUP BY department_id;
```

❖ **References:**

- "Database System Concepts", Avi Silberschatz, Henry F. Korth, S. Sudarshan, McGraw-Hill.
- "Database Management Systems", Raghu Ramakrishnan, Johannes Gehrke, McGraw-Hill.
- "Fundamentals of Database Systems", R. Elmasri, S. B. Navathe, Pearson.
- Oracle SQL Resources
- Other Internet Sources