COMP 5120

Jeff Ku

Caleb St.Germain

Homework 2

1.

- A foreign key constraint is a database constraint that ensures the consistency and integrity of data by linking two tables based on the values of their columns. It establishes a relationship between a column or a set of columns in one table (called the child table) and a primary key column or a unique key column in another table (called the parent table). The foreign key constraint ensures that the values in the child table's referencing column(s) always correspond to the values in the parent table's referenced column(s). It prohibits inserting or updating a row in the child table if the corresponding value in the parent table is missing or changed, and it also restricts deleting a row in the parent table if there are still dependent rows in the child table.
- Foreign key constraints are important because they help maintain data integrity and consistency in the database. They prevent orphaned rows (i.e., rows in the child table that do not have a corresponding row in the parent table) and redundant data. They also facilitate efficient querying and joining of tables by establishing relationships between them.
- Referential integrity is the property of a database that ensures that the relationships between tables are valid and consistent. It implies that every foreign key value in the child table must match a primary or unique key value in the parent table, and that every referenced key value in the parent table must exist and be unique. Referential integrity can be enforced by using foreign key constraints, which prevent any violation of the relationship rules.

2.

- In database design, a schema is a logical structure that defines the organization and relationships of the data elements in a database. There are three main types of schemas: external, internal, and conceptual.
- An external schema (also called a user schema or a view) describes the logical view of the database as seen by the application programs or the end-users. It specifies the subset of the database that is relevant to a particular user or group of users and hides the complexity of the underlying data model. An external schema can be different for each user or application and can provide customized access paths, data formats, and security controls.
- An internal schema (also called a physical schema or a storage schema) describes the physical organization of the database on the storage devices, such as the disk blocks, files, and indexes. It specifies the data structures, access methods, and storage allocation techniques used by the database management system (DBMS)

to store and retrieve data efficiently. An internal schema is typically hidden from the users and application programs.

- A conceptual schema (also called a global schema or a data model) describes the overall logical structure of the database as a set of interrelated entities, attributes, and relationships. It defines the data elements and their relationships in a database-independent and user-friendly way, using a high-level data model such as the entity-relationship (ER) model or the object-oriented data model. A conceptual schema provides a conceptual framework for designing and implementing the database and serves as a basis for deriving the external and internal schemas.
- The different schema layers are related to the concepts of logical and physical data independence, which are important in database design and maintenance. Logical data independence refers to the ability to change the conceptual schema without affecting the external schema or the application programs that use it. It means that the logical structure of the data can be modified or extended without requiring changes to the user views or the application code.
- Physical data independence refers to the ability to change the internal schema without affecting the conceptual schema or the external schema. It means that the physical storage and access methods of the data can be optimized or upgraded without affecting the logical structure or the user views of the data. The use of different schema layers enables logical and physical data independence by providing different levels of abstraction and encapsulation of the data elements and their relationships

3.

a)

```
SELECT DISTINCT pname
FROM Parts
INNER JOIN Catalog ON Parts.pid = Catalog.pid;
```

b)

```
SELECT Parts.pname, Suppliers.sname
FROM Parts
INNER JOIN Catalog ON Parts.pid = Catalog.pid
INNER JOIN Suppliers ON Catalog.sid = Suppliers.sid
WHERE Catalog.cost = (
  SELECT MAX(cost)
  FROM Catalog
  WHERE Catalog.pid = Parts.pid
);
```

c)

```
SELECT DISTINCT sid
FROM Catalog
WHERE pid IN (
  SELECT pid
  FROM Parts
  WHERE color = 'red'
)
GROUP BY sid
HAVING COUNT(DISTINCT pid) = (
  SELECT COUNT(*)
  FROM Parts
  WHERE color = 'red'
);
```

d)

```
SELECT sname
FROM Suppliers
WHERE NOT EXISTS (
  SELECT pid
  FROM Parts
  WHERE NOT EXISTS (
    SELECT sid
    FROM Catalog
    WHERE Catalog.sid = Suppliers.sid AND Catalog.pid = Parts.pid
  )
);
```

4.

a) To find the names of all employees who work for Auburn Bank, we can use a selection operation to select rows from the Works table where the company-name is "Auburn Bank", and then use a projection operation to return only the person-name attribute:

π person-name (σ company-name = ‘Auburn Bank’ (Works))
π person-name (σ company-name = ‘Auburn Bank’ (Works))

b) To find the names and cities of residence of all employees who work for Auburn Bank, we can use a join operation to combine the Works and Employee tables on the person-name attribute, and then use a selection operation to select rows where the company-name is "Auburn Bank". Finally, we can use a projection operation to return only the person-name and city attributes from the resulting table:

π person-name, city (σ company-name = 'Auburn Bank' (Works ⋈ Employee))

c) To find the names, street address, and cities of residence of all employees who work for Auburn Bank and earn more than $50,000 per year, we can use a join operation to combine the Works and Employee tables on the person-name attribute, and then use a selection operation to select rows where the company-name is "Auburn Bank" and the salary is greater than $50,000. Finally, we can use a projection operation to return the person-name, street, and city attributes from the resulting table:

   π person-name, street, city (σ company-name = 'Auburn Bank' $\wedge$ salary > 50000 (Works $\bowtie$ Employee))

d) To find the names of all employees in this database who live in the same city as the company for which they work, we can use two join operations. First, we join the Works and Company tables on the company-name attribute to get a table that lists all employees and their company's city. Then, we join this table with the Employee table on the city attribute to get a table that lists all employees who live in the same city as their company. Finally, we can use a projection operation to return only the person-name attribute from the resulting table:

   π person-name (Works $\bowtie$ Company) $\bowtie$ Employee where city = Company.city