Campus Corners
Scratch2Corporate...

DBMS

NIKHIL B. DETHE

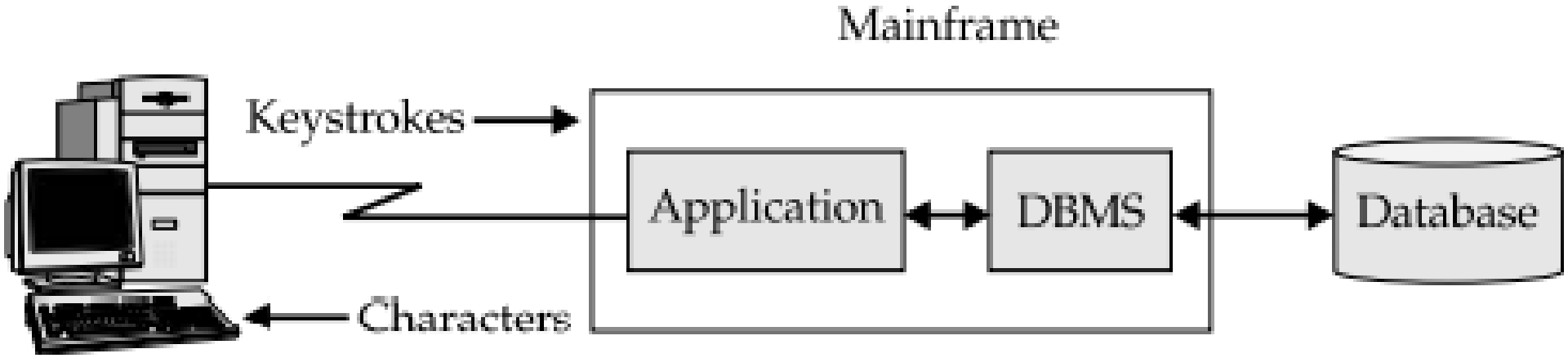
DATA SCIENTIST

SQL

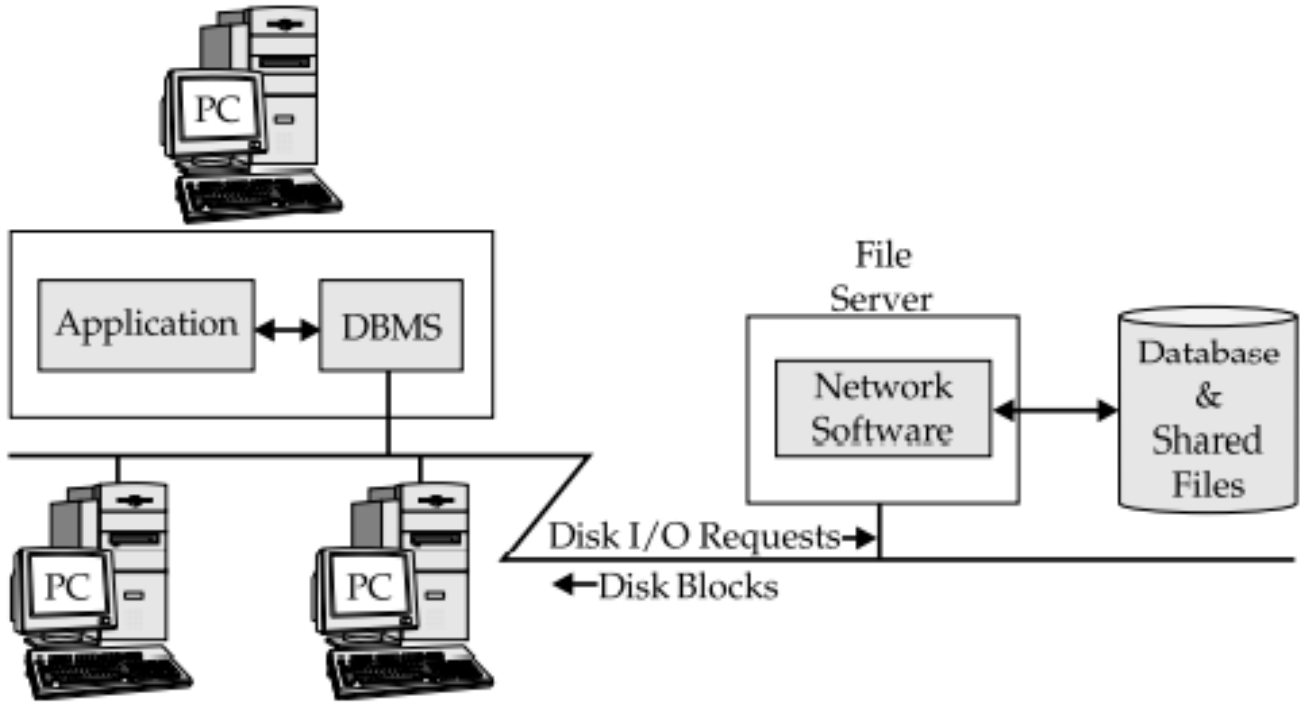
- The important information stored on computer system is called as **database**.
- The computer program that controls the database is called a **database management system (DBMS)**.
- SQL means **“Structured Query Language”**, it is also called as ***SEQUEL*.**
- SQL is a tool for **organizing, managing**, and **retrieving data** stored by a computer database.
- SQL is a computer language that you use to **interact** with a database.
- SQL works with one specific type of database, called a **relational database**.
- When you need to retrieve data from a database, you use the SQL to make the request, then DBMS processes the SQL request, retrieves the requested data, and returns it to you.
- This whole process of requesting data from a database and receiving the results is called a database query.

Functions of SQL

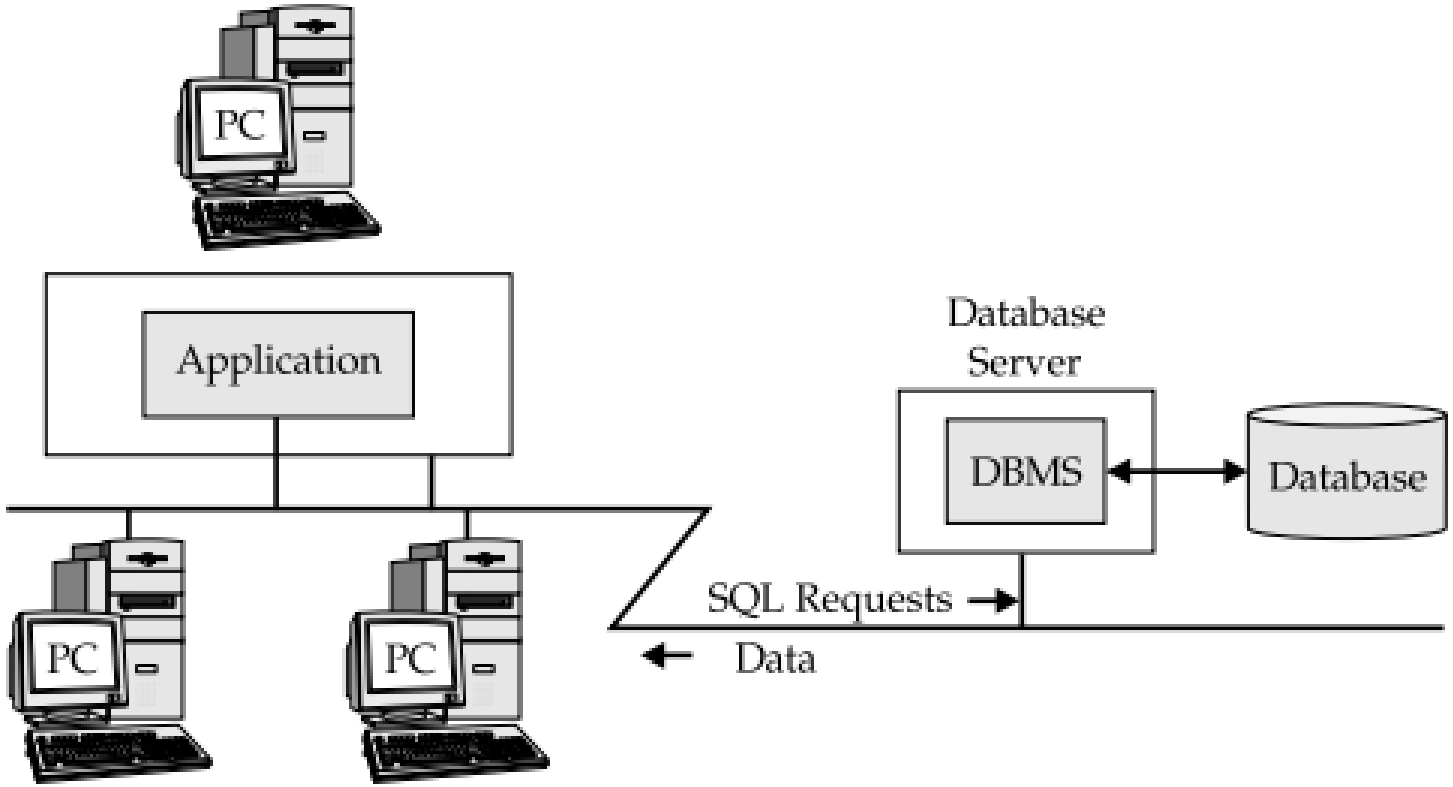
- **Data definition** SQL lets a user define the structure and organization of the stored data and relationships among the stored data items.
- **Data retrieval** SQL allows a user or an application program to retrieve stored data from the database and use it.
- **Data manipulation** SQL allows a user or an application program to update the database by adding new data, removing old data, and modifying previously stored data.
- **Access control** SQL can be used to restrict a user's ability to retrieve, add, and modify data, protecting stored data against unauthorized access.
- **Data sharing** SQL is used to coordinate data sharing by concurrent users, ensuring that changes made by one user do not inadvertently wipe out changes made at nearly the same time by another user.
- **Data integrity** SQL defines integrity constraints in the database, protecting it from corruption due to inconsistent updates or system failures.



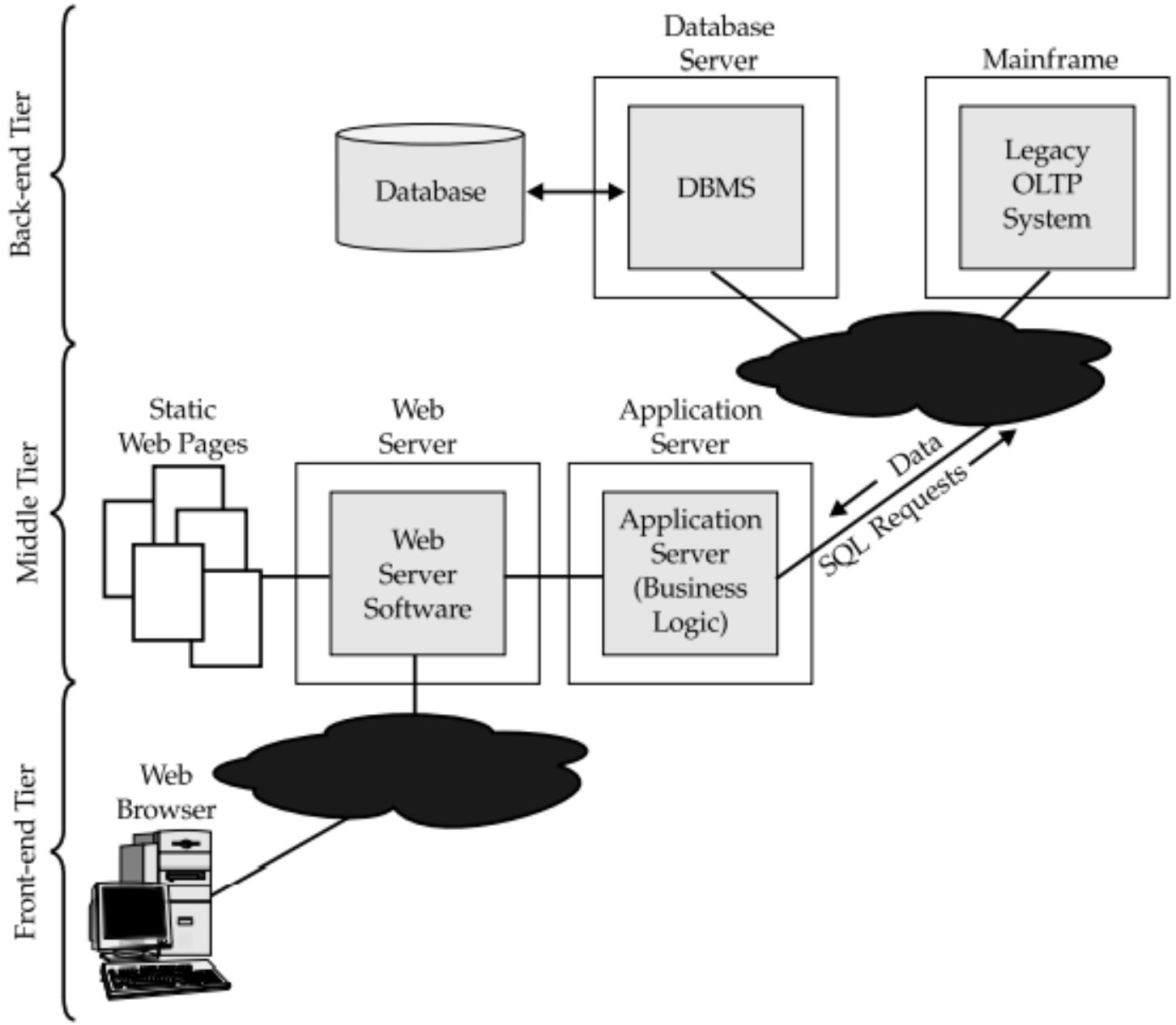
Centralized Architecture



File Server Architecture



Client/Server Architecture



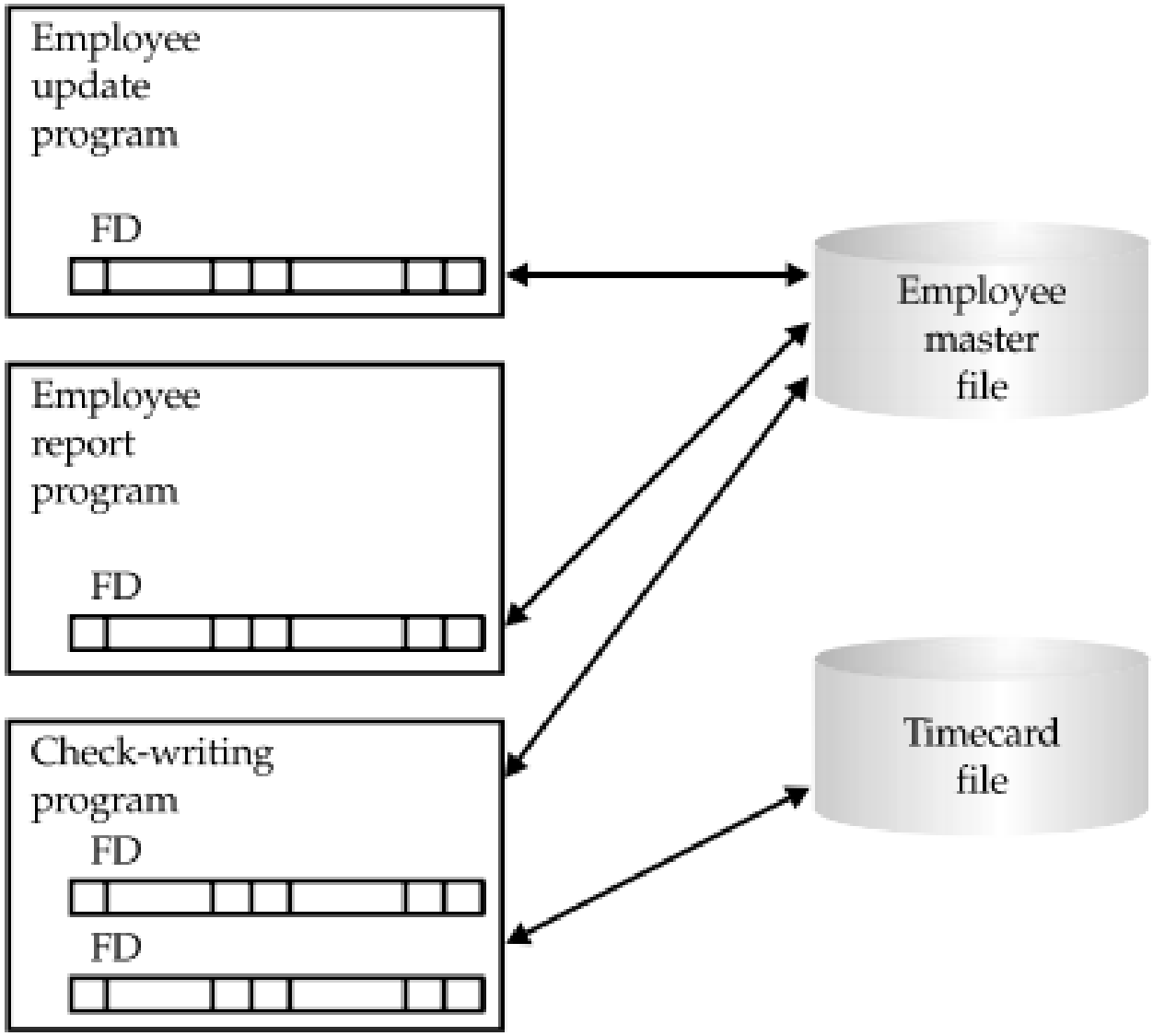
Multitier Architecture

Early Data Models

- File Management Systems
- Hierarchical Databases
- Network Databases
- Relational Data Model

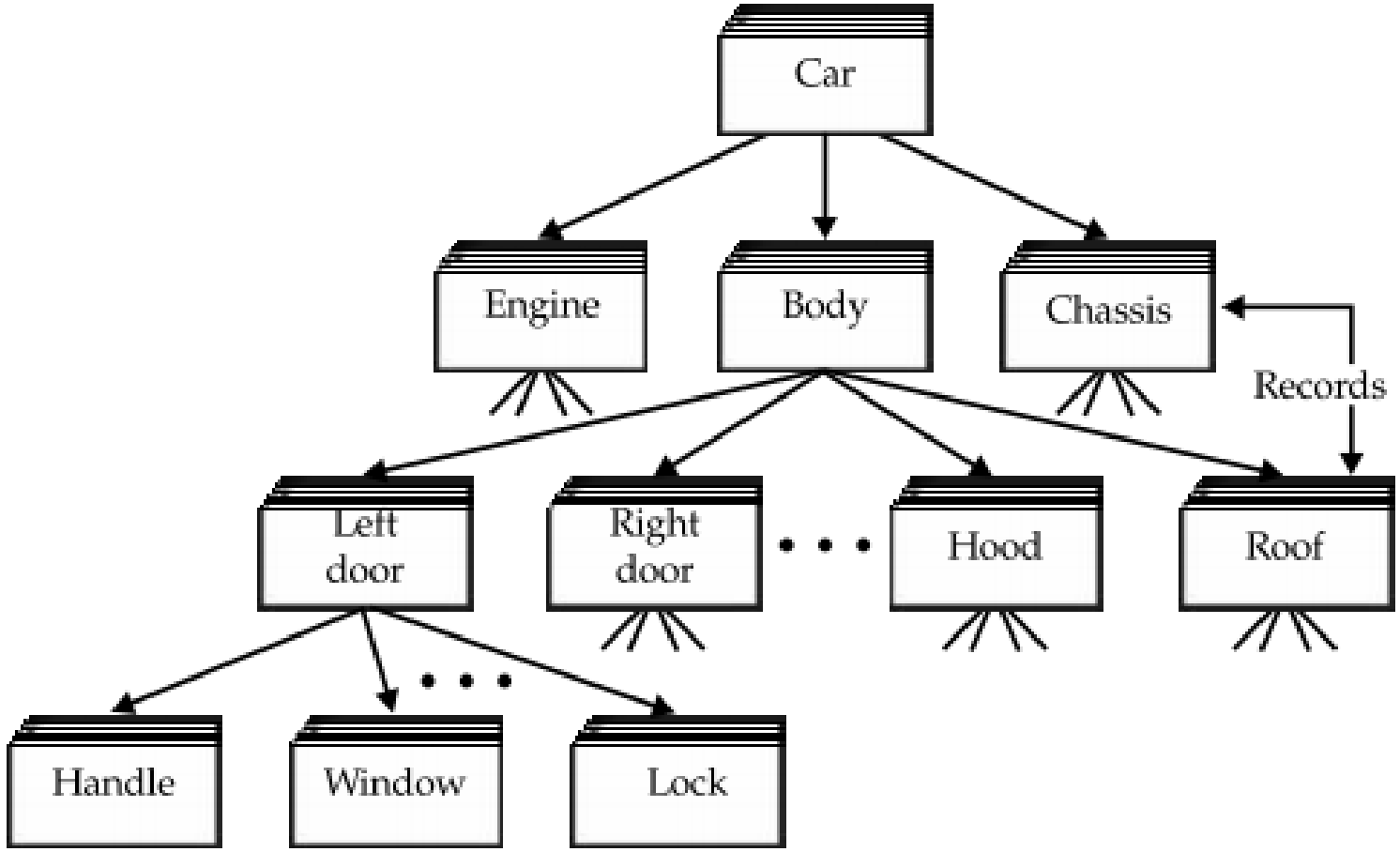
File Systems

1) Data Redundancy
2) Data Inconsistency
3) Difficulty in Accessing Data
4) Limited Data Sharing
5) Integrity Problems
6) Atomicity Problems
7) Concurrent Access Anomalies
8) Security Problems



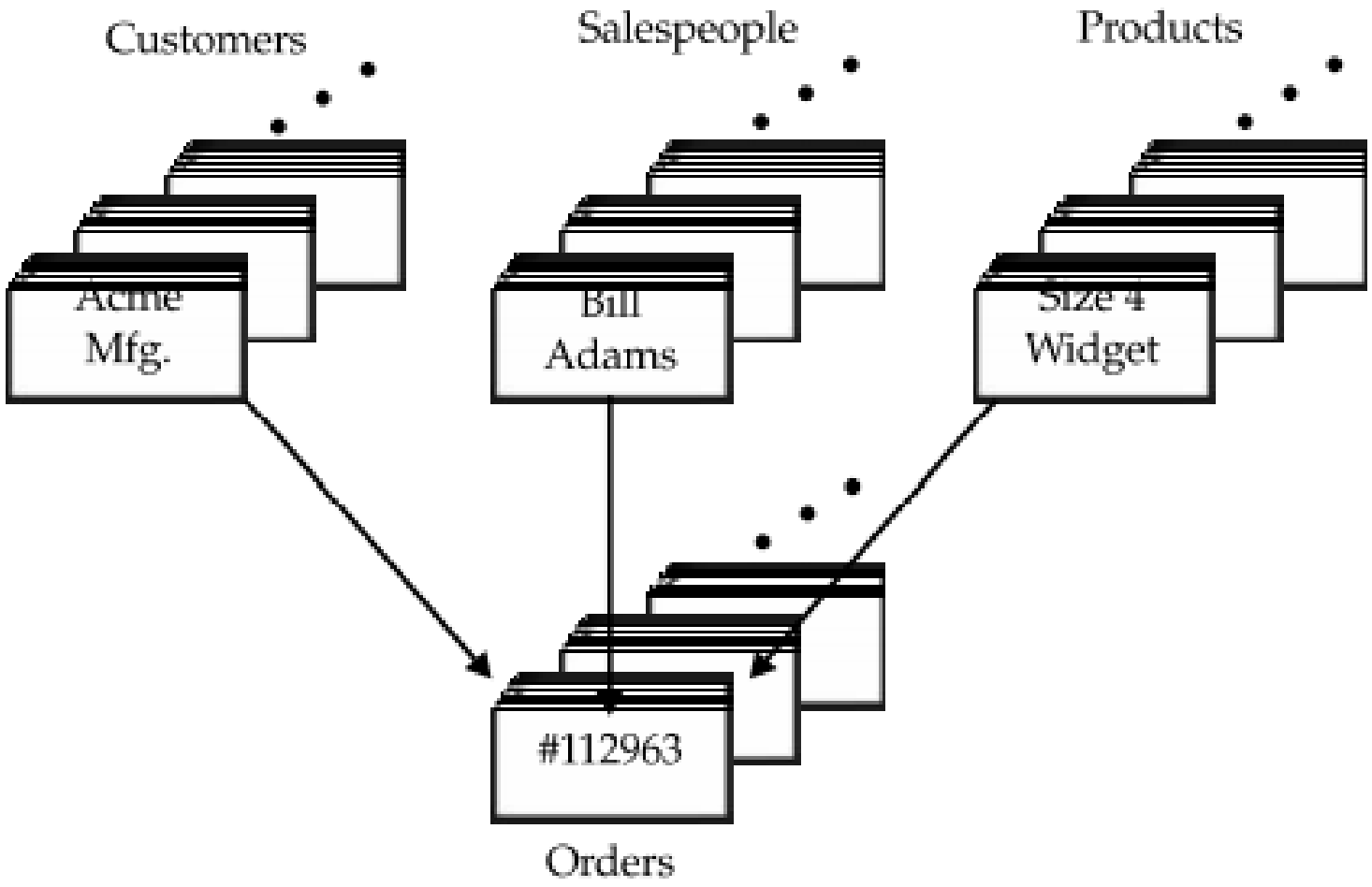
Hierarchical Databases

1) Rigid structure
2) One parent per child
3) Navigation system is complex
4) Data must be organized in a hierarchical way
5) Lack structural independence
6) Many too many relationships not supported
7) Data independence
8) Lack of standards
9) Poor flexibility



Network Databases

1) Data relationships must be predefined
2) Rigid structure
3) Much more complex than the hierarchical date model
4) Users are still required to know the physical representation of the database
5) Information can be related in various and complicated ways
6) Lack structural independence



Relational Data Model

1) Structured independence is promoted
2) Users do not have to know the physical representation of the database
3) Use of SQL language to access data
4) Easier database design
5) Tabular view improves simplicity
6) Support large amounts of data
7) Data independence
8) Multi-level relationships between data sets
9) No need to predefined data relationships

PRODUCTS Table

DESCRIPTION	PRICE	QTY_ON_HAND
Size 3 Widget	$107.00	207
Size 4 Widget	$117.00	139
Hinge Pin	$350.00	14
.		
.		
.		

ORDERS Table

ORDER_NUM	COMPANY	PRODUCT	QTY
112963	Acme Mfg.	41004	28
112975	JCP Inc.	2A44G	6
112983	Acme Mfg.	41004	6
113012	JCP Inc.	41003	35
.			
.			
.			

CUSTOMERS Table

COMPANY	CUST_REP	CREDIT_LIMIT
Acme Mfg.	105	$50,000.00
JCP Inc.	103	$50,000.00
.		
.		
.		

Codd's 12 Rules

for Relational Databases

1. **Information rule:** All information in a relational database is represented explicitly at the logical level and in exactly one way—by values in tables.
2. **Guaranteed access rule:** Each and every datum (atomic value) in a relational database is guaranteed to be logically accessible by resorting to a combination of table name, primary key value, and column name.
3. **Systematic treatment of NULL values:** NULL values (distinct from an empty character string or a string of blank characters and distinct from zero or any other number) are supported in a fully relational DBMS for representing missing information and inapplicable information in a systematic way, independent of the data type.
4. **Dynamic online catalog based on the relational model:** The database description is represented at the logical level in the same way as ordinary data, so that authorized users can apply the same relational language to its interrogation as they apply to the regular data.
5. **Comprehensive data sublanguage rule:** A relational system may support several languages and various modes of terminal use (for example, the fill-in-the-blanks mode). However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings.

Codd's 12 Rules

for Relational Databases

6. **View updating rule:** All views that are theoretically updateable are also updateable by the system.
7. **High-level insert, update, and delete:** The capability of handling a base relation or a derived relation as a single operand applies not only to the retrieval of data, but also to the insertion, update, and deletion of data.
8. **Physical data independence:** Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representations or access methods.
9. **Logical data independence:** Application programs and terminal activities remain logically unimpaired when information-preserving changes of any kind that theoretically permit unimpairment are made to the base tables.
10. **Integrity independence:** Integrity constraints specific to a particular relational database must be definable in the relational data sublanguage and storable in the catalog, not in the application programs.
11. **Distribution independence:** A relational DBMS has distribution independence.
12. **Nonsubversion rule:** If a relational system has a low-level (single record at a time) language, that low level cannot be used to subvert or bypass the integrity rules and constraints expressed in the higher-level relational language (multiple records at a time).

Characteristics

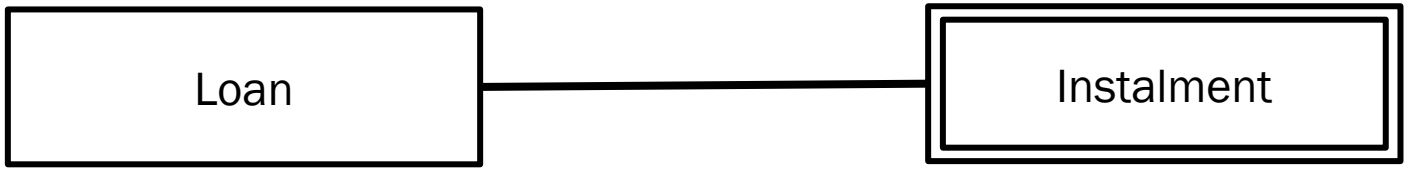
of Relational Databases

1. Real-world entity
2. Relation-based tables
3. Isolation of data and application
4. Less redundancy
5. Consistency
6. Query Language
7. ACID Properties
8. Multiuser and Concurrent Access
9. Multiple views
10. Security

ER-Model

basic Concept

1) **Entity:** An entity may be any object, class, person or place. Entities are represented by means of rectangles. Rectangles are named with the entity set they represent.
   a) **Weak Entity :** An entity that depends on another entity called a weak entity. The weak entity doesn't contain any key attribute of its own. The weak entity is represented by a double rectangle.



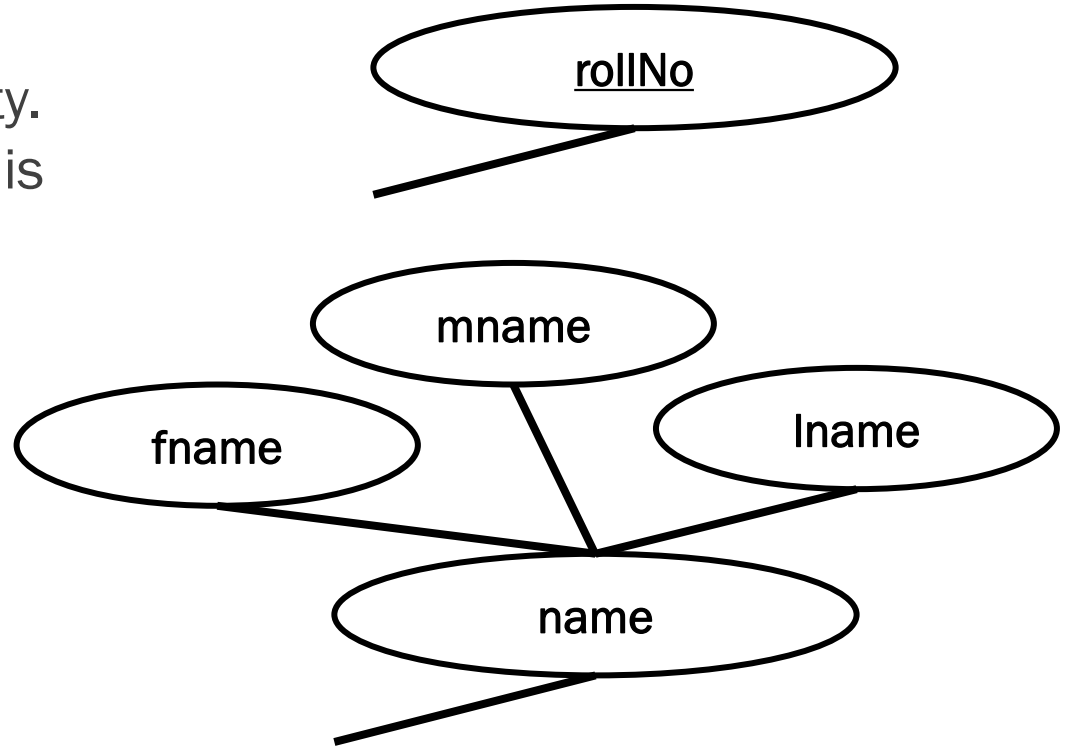
ER-Model

basic Concept

2) **Attribute:** Attributes are the properties of entities. Attributes are represented by means of ellipses. Every ellipse represents one attribute and is directly connected to its entity.

   a) **Key Attribute:** The key attribute is used to represent the main characteristics of an entity. It represents a primary key. The key attribute is represented by an ellipse with the text underlined.

   b) **Composite Attribute:** An attribute that composed of many other attributes is known as a composite attribute. The composite attribute is represented by an ellipse, and those ellipses are connected with an ellipse.

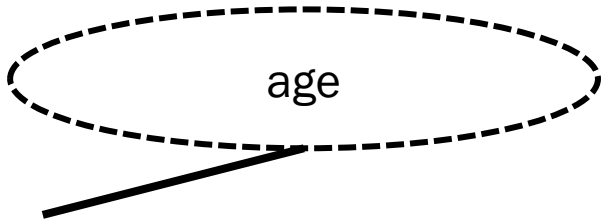


ER-Model

basic Concept

c) **Multivalued Attribute:** An attribute can have more than one value. These attributes are known as a multivalued attribute. The double oval is used to represent multivalued attribute.

d) **Derived Attribute:** An attribute that can be derived from another attribute is known as a derived attribute. It can be represented by a dashed ellipse.

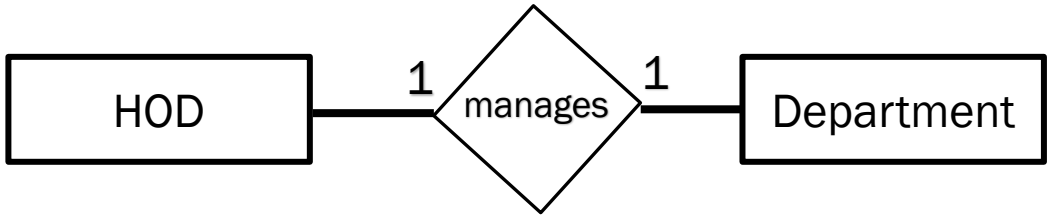


ER-Model

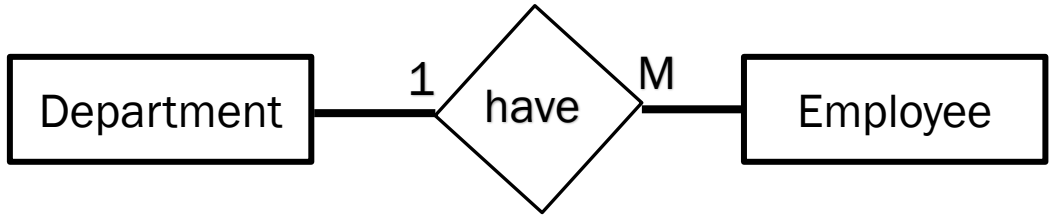
basic Concept

3) **Relationship:** A relationship is used to describe the relation between entities. Diamond or rhombus is used to represent the relationship.

a) **One-to-One Relationship:** When only one instance of an entity is associated with the relationship, then it is known as one-to-one relationship.



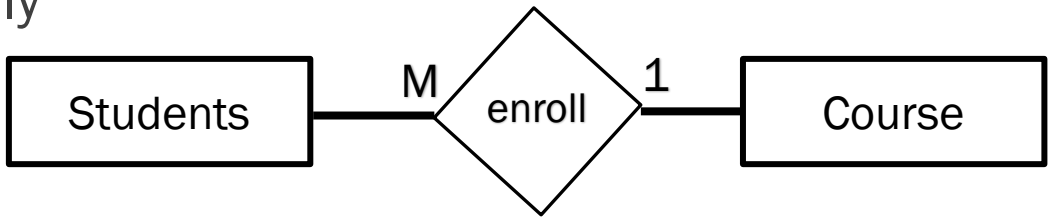
b) **One-to-many relationship:** When only one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then this is known as a a one-to-many relationship.



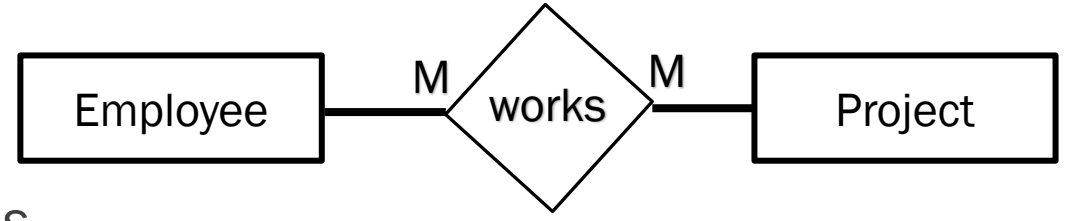
ER-Model

basic Concept

c) **Many-to-one relationship:** When more than one instance of the entity on the left, and only one instance of an entity on the right associates with the relationship then it is known as a many-to-one relationship.



d) **Many-to-many relationship:** When more than one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then it is known as a many-to-many relationship.



Types of Keys

in Relational Databases

1. **Primary key:** A primary is a column or set of columns in a table that uniquely identifies tuples (rows) in that table.
2. **Super key:** A super key is a set of one of more columns (attributes) to uniquely identify rows in a table.
3. **Candidate key:** A super key is a minimal super key with no redundant attribute is known as candidate key.
4. **Alternate key:** Out of all candidate keys, only one gets selected as primary key, remaining keys are known as alternate or secondary keys.
5. **Composite key:** A key that consists of more than one attribute to uniquely identify rows (also known as records & tuples) in a table is called composite key.
6. **Foreign key:** Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables.

Keys

example

Employee

EmpNumber	EmpCode	Name	Department	Age	Address	ContactNo
E-10001	RD1001	Nick	R n D	33	New York	+1468888
E-10002	MK2001	Nancy	Marketing	25	London	+2066766
E-10003	FA3001	Dakota	Finance	29	New York	+1948548
E-10004	RD1002	Sophie	R n D	32	New York	+1884690
E-10005	MK2002	Jenny	Marketing	44	London	+2066666

Super Keys

(EmpCode)

(EmpNumber)

(EmpCode, EmpNumber)

(EmpCode, Name)

(EmpNumber, Name)

(EmpCode, EmpNumber, Name)

Candidate Keys

(EmpCode)

(EmpNumber)

Primary Key

(EmpNumber)

Alternate Key

(EmpCode)

Composite Keys

(EmpCode)

(EmpNumber)

(Name)

Data Integrity

To **preserve** the **consistency** and **correctness** of its stored data, a relational DBMS typically **imposes** one or more data **integrity constraints**. These constraints restrict the data values that can be inserted into the database or that result from a database update.

Integrity Constraints

1. **Required data:** Some columns in a database must contain a valid data value in every row; they are not allowed to contain missing or NULL values.
2. **Validity checking:** Every column in a database has a domain, a set of data values that are legal for that column.
3. **Entity integrity:** The primary key of a table must contain a unique value in each row, which is different from the values in all other rows.
4. **Referential integrity:** A foreign key in a relational database links each row in the child table containing the foreign key to the row of the parent table containing the matching primary key value.
5. **Other data relationships:** The real-world situation modeled by a database will often have additional constraints that govern the legal data values that may appear in the database.
6. **Business rules:** Updates to a database may be constrained by business rules governing the real-world transactions that are represented by the updates.
7. **Consistency:** Many real-world transactions cause multiple updates to a database.

Data Representation in RDB

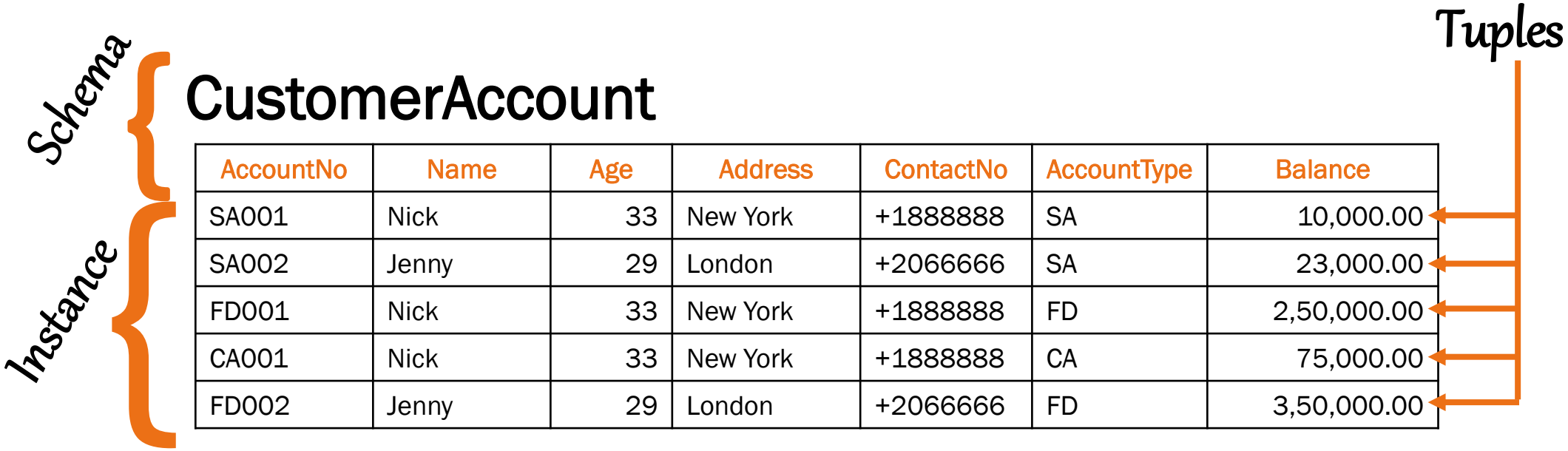
CustomerAccount

rows

AccountNo	Name	Age	Address	ContactNo	AccountType	Balance
SA001	Nick	33	New York	+1888888	SA	10,000.00
SA002	Jenny	29	London	+2066666	SA	23,000.00
FD001	Nick	33	New York	+1888888	FD	2,50,000.00
CA001	Nick	33	New York	+1888888	CA	75,000.00
FD002	Jenny	29	London	+2066666	FD	3,50,000.00

columns

Data Representation in RDB

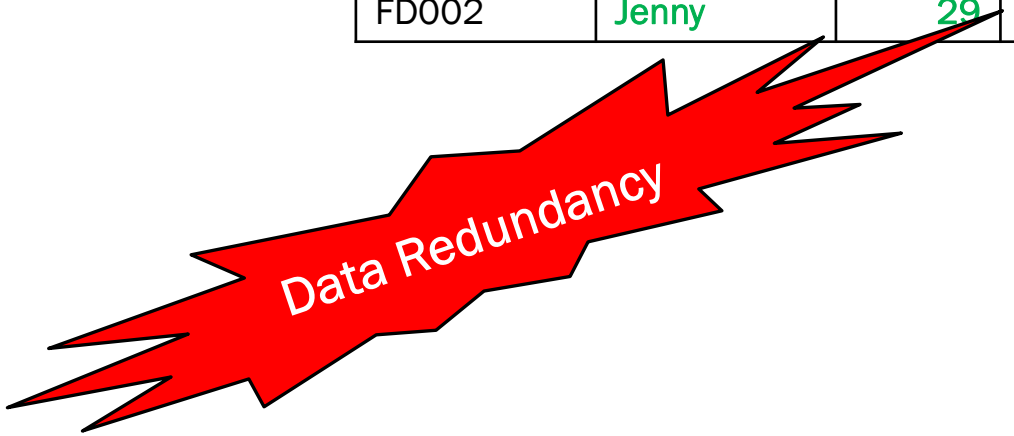


AccountNo	Name	Age	Address	ContactNo	AccountType	Balance
SA001	Nick	33	New York	+1888888	SA	10,000.00
SA002	Jenny	29	London	+2066666	SA	23,000.00
FD001	Nick	33	New York	+1888888	FD	2,50,000.00
CA001	Nick	33	New York	+1888888	CA	75,000.00
FD002	Jenny	29	London	+2066666	FD	3,50,000.00

Data is not in a Normal Form

CustomerAccount

AccountNo	Name	Age	Address	ContactNo	AccountType	Balance
SA001	Nick	33	New York	+1888888	SA	10,000.00
SA002	Jenny	29	London	+2066666	SA	23,000.00
FD001	Nick	33	New York	+1888888	FD	2,50,000.00
CA001	Nick	33	New York	+1888888	CA	75,000.00
FD002	Jenny	29	London	+2066666	FD	3,50,000.00



Data redundancy may lead to some serious issues

1) High memory storage capacity
2) Inconsistent data
3) May lead to insert, update & delete anomalies.

Logically related data separated

Name	Age	Address	ContactNo
Nick	33	New York	+1888888
Jenny	29	London	+2066666

Customer

AccountNo	AccountType	Balance
SA001	SA	10,000.00
SA002	SA	23,000.00
FD001	FD	2,50,000.00
CA001	CA	75,000.00
FD002	FD	3,50,000.00

Account

Data in a Normal form

pk

CustomerId	Name	Age	Address	ContactNo
C0001	Nick	33	New York	+1888888
C0002	Jenny	29	London	+2066666

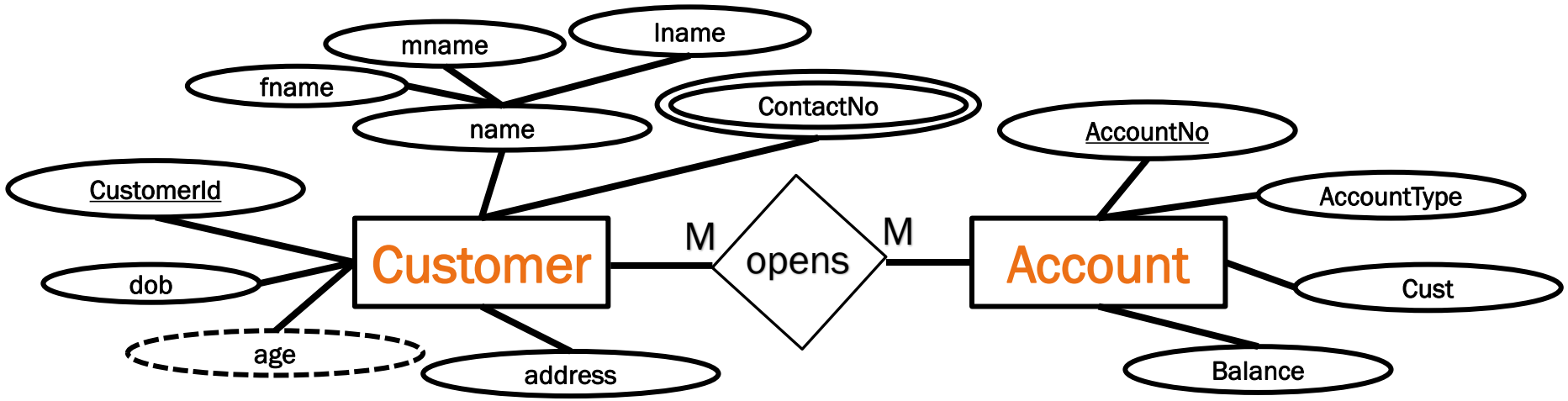
Customer

pk fk

AccountNo	AccountType	Cust	Balance
SA001	SA	*C0001*	10,000.00
SA002	SA	*C0002*	23,000.00
FD001	FD	*C0001*	2,50,000.00
CA001	CA	*C0001*	75,000.00
FD002	FD	*C0002*	3,50,000.00

Account

ER-Representation



SQL Commands

in Relational Databases

1. **Data Definition Language (DDL):** DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc. All the command of DDL are auto-committed that means it permanently save all the changes in the database. **(CREATE, ALTER, DROP, TRUNCATE)**
2. **Data Manipulation Language (DML):** DML commands are used to modify the database. It is responsible for all form of changes in the database. The command of DML is not auto-committed. They can be rollback. **(INSERT, UPDATE, DELETE)**
3. **Data Query Language (DQL):** DQL is used to fetch the data from the database. **(SELECT)**
4. **Data Control Language (DCL):** DCL commands are used to grant and take back authority from any database user. **(GRANT, REVOKE)**
5. **Transaction Control Language (TCL):** TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only. These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them. **(COMMIT, ROLLBACK, SAVEPOINT)**

DDL (Data Definition Language)

SQL Commands

- **Create**

```
CREATE TABLE Students   ( RollNo INT PRIMARY KEY,
        Name VARCHAR(20), Marks NUMERIC(5,2) );
```

- **Alter**

```
ALTER TABLE Students
ADD DateOfBirth DATE;
```

- **Drop**

```
DROP TABLE Students;
```

- **Truncate**

```
TRUNCATE TABLE Students;
```

DML (Data Manipulation Language)

SQL Commands

- **Insert**

```
INSERT INTO Students (RollNo, Name, Marks) VALUES (1, 'Nick', 33);
```

- **Update**

```
UPDATE Students
SET Name = 'Nick Carter', Marks = 99
WHERE RollNo = 1;
```

- **Delete**

```
DELETE FROM Students
WHERE RollNo = 1
```

DQL (Data Query Language)

SQL Commands

- **Select**

```
SELECT RollNo, Name, Marks

FROM Students

WHERE RollNo = 1;
```

OR

```
SELECT *

FROM Students;
```

DCL (Data Control Language)

SQL Commands

- **Grant**

```
GRANT SELECT, UPDATE
ON Students
TO AdminStaff, TeachingStaff;
```

```
GRANT ALL
ON Students
TO PUBLIC;
```

- **Revoke**

```
REVOKE SELECT, UPDATE
ON Students
FROM  AdminStaff, TeachingStaff;
```

```
REVOKE ALL
ON Students
FROM  PUBLIC;
```

TCL (Transaction Control Language)

SQL Commands

- **Commit**

```
COMMIT;
```

- **Rollback**

```
ROLLBACK;
```

- **Savepoint**

```
SAVEPOINT sp_2;
```

Sample Schema

for Practice

Primary Key
Foreign Key

SALESREPS Table

EMPL_NUM	NAME	AGE	REP_OFFICE	TITLE	HIRE_DATE	MANAGER	QUOTA	SALES
105	Bill Adams	37	13	Sales Rep	2006-02-12	104	$350,000.00	$367,911.00
109	Mary Jones	31	11	Sales Rep	2007-10-12	106	$300,000.00	$392.725.00
102	Sue Smith	48	21	Sales Rep	2004-12-10	108	$350,000.00	$474,050.00

ORDERS Table

ORDER_NUM	ORDER_DATE	CUST	REP	MFR	PRODUCT	QTY	AMOUNT
112961	2007-12-17	2117	106	REI	2A44L	7	$31,500.00
113012	2008-01-11	2111	105	ACI	41003	35	$3,745.00
112989	2008-01-03	2101	106	FEA	114X	6	$1,458.00

OFFICES Table

OFFICE	CITY	REGION	MGR	TARGET	SALES
22	Denver	Western	108	$300,000.00	$186,042.00
11	New York	Eastern	106	$575,000.00	$692,637.00
12	Chicago	Eastern	104	$800,000.00	$735,042.00

PRODUCTS Table

MFR_ID	PRODUCT_ID	DESCRIPTION	PRICE	QTY_ON_HAND
REI	2A45C	Ratchet Link	$79.00	210
ACI	4100Y	Widget Remover	$2,750.00	25
QSA	Xk47	Reducer	$355.00	38

CUSTOMERS Table

CUST_NUM	COMPANY	CUST_REP	CREDIT_LIMIT
2111	JCP Inc.	103	$50,000.00
2102	First Corp.	101	$65,000.00
2103	Acme Mfg.	105	$50,000.00

Sample Schema

for Practice

Primary Key

Foreign Key

SALESREPS Table

EMPL_NUM	NAME	AGE	REP_OFFICE	TITLE	HIRE_DATE	MANAGER	QUOTA	SALES
105	Bill Adams	37	13	Sales Rep	2006-02-12	104	$350,000.00	$367,911.00
109	Mary Jones	31	11	Sales Rep	2007-10-12	106	$300,000.00	$392.725.00
102	Sue Smith	48	21	Sales Rep	2004-12-10	108	$350,000.00	$474,050.00

ORDERS Table

ORDER_NUM	ORDER_DATE	CUST	REP	MFR	PRODUCT	QTY	AMOUNT
112961	2007-12-17	2117	106	REI	2A44L	7	$31,500.00
113012	2008-01-11	2111	105	ACI	41003	35	$3,745.00
112989	2008-01-03	2101	106	FEA	114X	6	$1,458.00

OFFICES Table

OFFICE	CITY	REGION	MGR	TARGET	SALES
22	Denver	Western	108	$300,000.00	$186,042.00
11	New York	Eastern	106	$575,000.00	$692,637.00
12	Chicago	Eastern	104	$800,000.00	$735,042.00

PRODUCTS Table

MFR_ID	PRODUCT_ID	DESCRIPTION	PRICE	QTY_ON_HAND
REI	2A45C	Ratchet Link	$79.00	210
ACI	4100Y	Widget Remover	$2,750.00	25
QSA	Xk47	Reducer	$355.00	38

CUSTOMERS Table

CUST_NUM	COMPANY	CUST_REP	CREDIT_LIMIT
2111	JCP Inc.	103	$50,000.00
2102	First Corp.	101	$65,000.00
2103	Acme Mfg.	105	$50,000.00

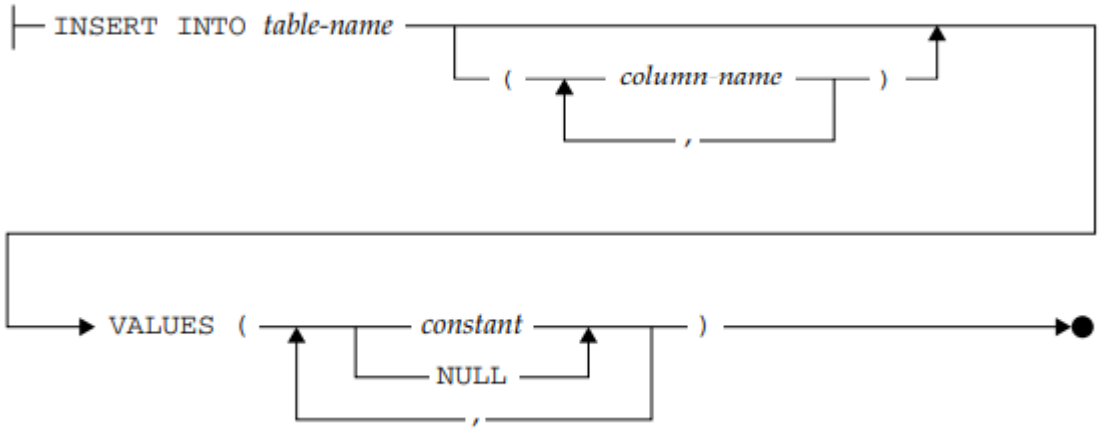
Insert Statement

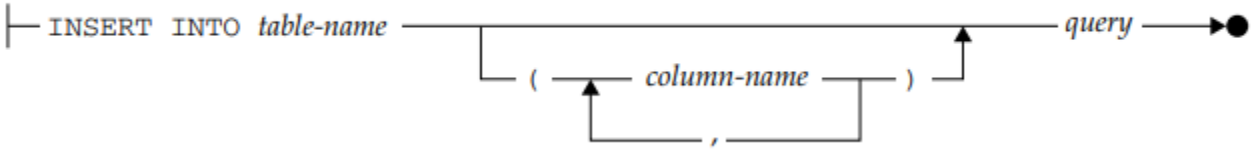
variations

- **Single-row INSERT:** A single-row INSERT statement adds a single new row of data to a table. It is commonly used in daily applications- for example, data entry programs.
- **Multirow INSERT:** A multirow INSERT statement extracts rows of data from another part of the database and adds them to a table. It is commonly used, for example in end-of-month processing when old rows of a table are moved to an inactive table, or when monthly results are summarized into a table that has been set up to hold them.
- **Bulk load:** A bulk load utility adds data to a table from a file that is outside of the database. It is commonly used to initially load the database or to incorporate data downloaded from another computer system or collected from many sites.

Insert Statement

in SQL

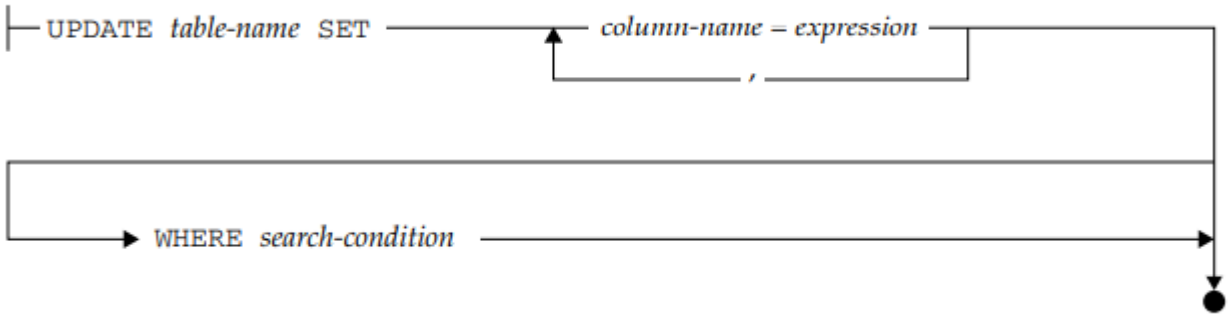




Update Statement

in SQL

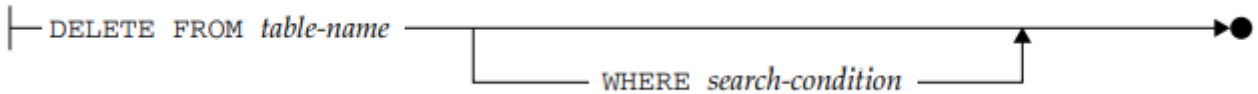
- The SQL UPDATE Query is used to modify the existing records in a table.
- You can use the WHERE clause with the UPDATE query to update the selected rows, otherwise all the rows would be affected.
- UPDATE is a DML Command so it can be rolled back.
- The UPDATE command returns the number of records that were updated by its execution



Delete Statement

in SQL

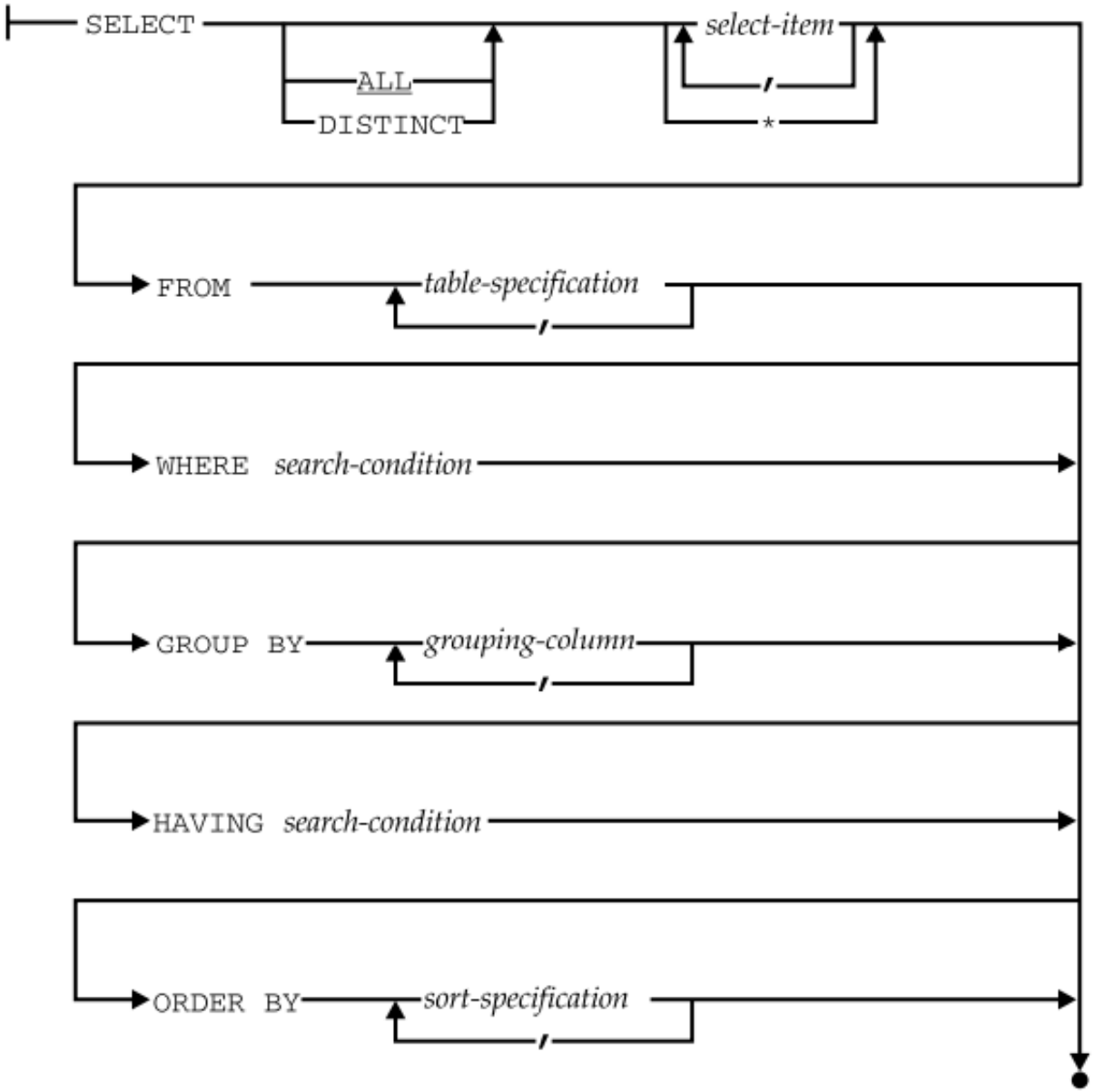
- The DELETE statement in SQL is a Data Manipulation Language(DML) Command.
- It is used to delete existing records from an existing table.
- We can delete a single record or multiple records depending on the condition specified in the query.
- The conditions are specified in the WHERE clause of the DELETE statement.
- If we omit the WHERE clause then all the records will be deleted, and the table will be empty.
- The DELETE statement scans every row before deleting it. Thus, it is slower as compared to TRUNCATE command.
- If we want to delete all the records of a table, it is preferable to use TRUNCATE in place of DELETE as the former is faster than the latter.
- DELETE is a DML Command so it can be rolled back.
- The DELETE command returns the number of records that were deleted by its execution
- We do not need to list fields in the DELETE statement since we are deleting the entire row from the table.



ACID Properties

in RDBMS

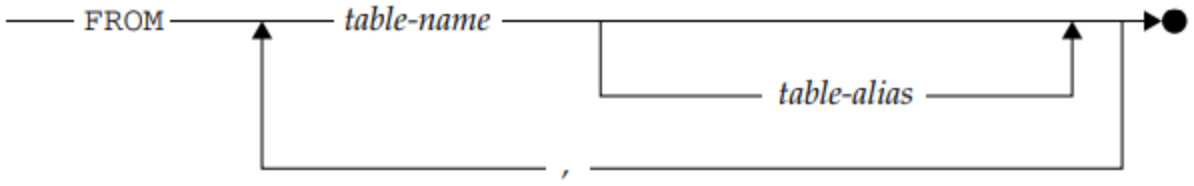
- **Atomic** A transaction has an all-or-nothing nature. Either all operations in a transaction are performed or none of them are performed. If some statements are executed and the transaction fails, the results of the statements that executed must be rolled back. Only when all statements are executed properly can a transaction be considered complete and the results of the transaction applied to the database.
- **Consistent** A transaction must transform the database from one consistent state to another. The database must be consistent at the end of each transaction, meaning that all rules that define and constrain the data must be adhered to before the transaction can end. Also, no user should see inconsistent data because of changes made by transactions that have not yet completed.
- **Isolated** Each transaction must execute on its own without interference from other transactions. To be isolated, no transaction can act upon changes made by other transactions until those transactions are complete.
- **Durable** Once a transaction is complete, all changes made by it should be preserved. The data should be in a consistent state, even if a hardware or application error occurs after completion of the transaction. In object-oriented programming, the term *persistence* is used for this property.



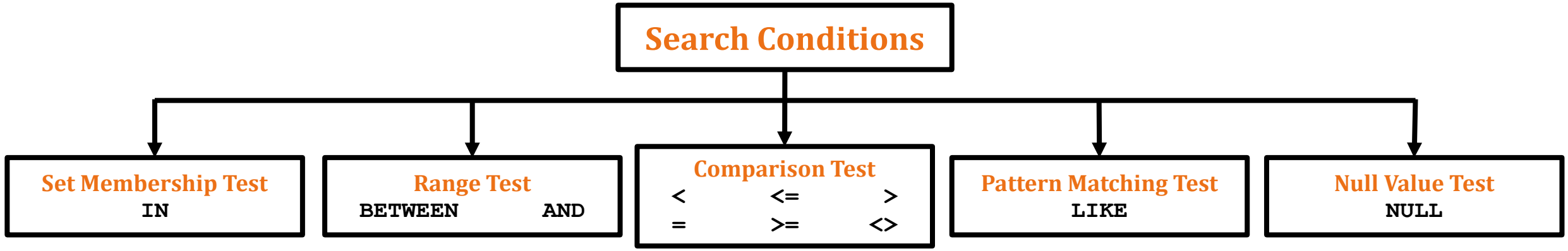
SELECT Statement

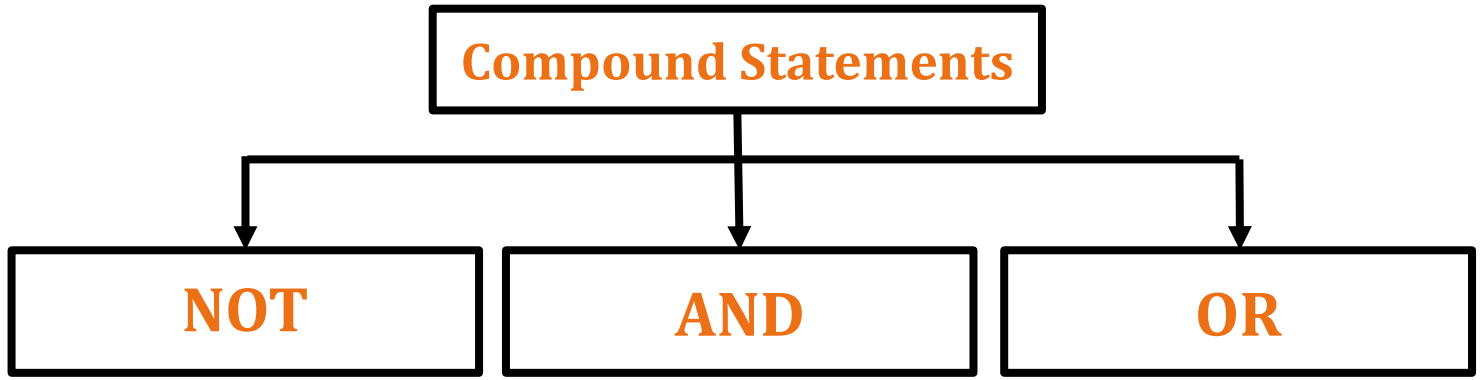
From Clause

in Relational Databases



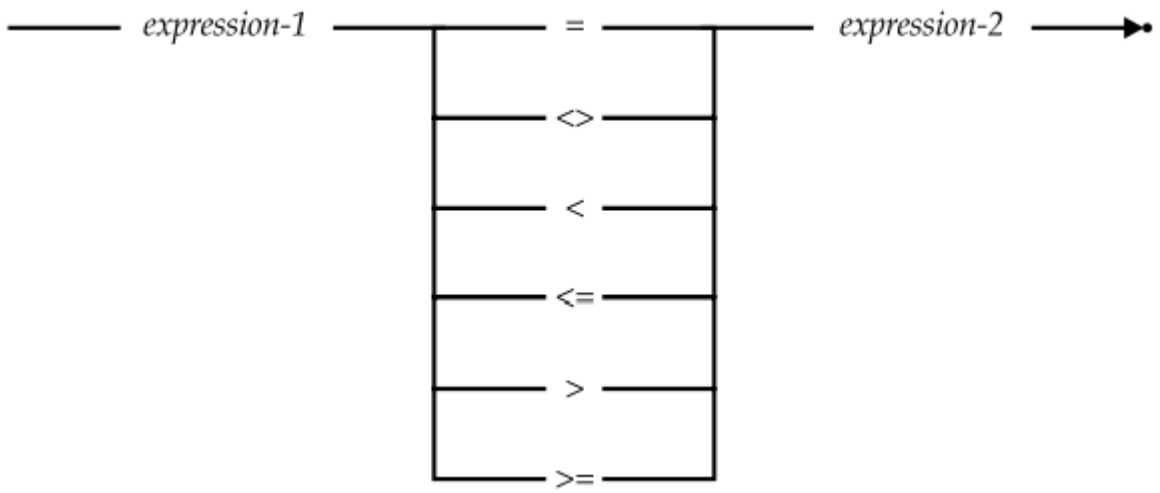
Search Conditions





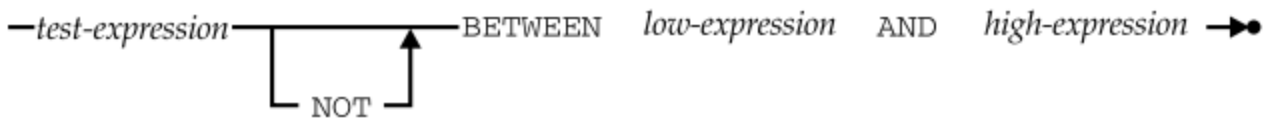
Comparison Test

Search Conditions in Where Clause



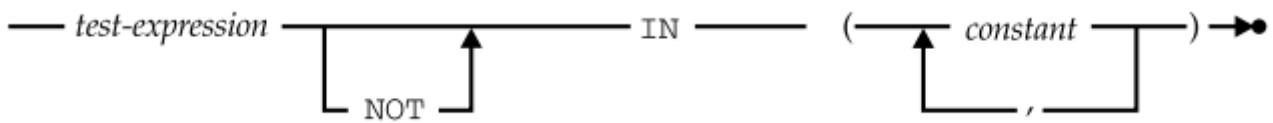
Range Test

Search Conditions in Where Clause



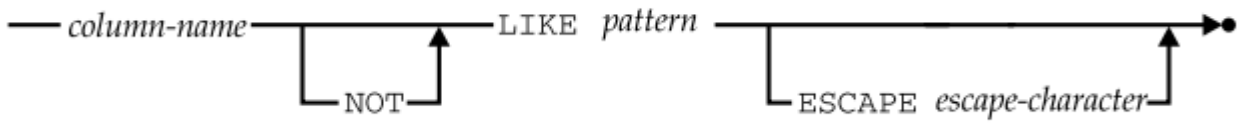
Set Membership Test

Search Conditions in Where Clause



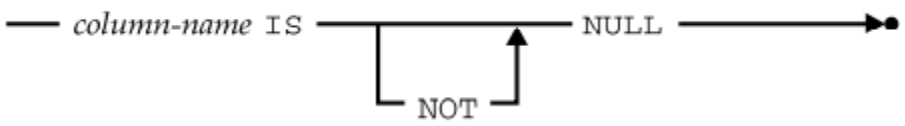
Pattern Matching Test

Search Conditions in Where Clause



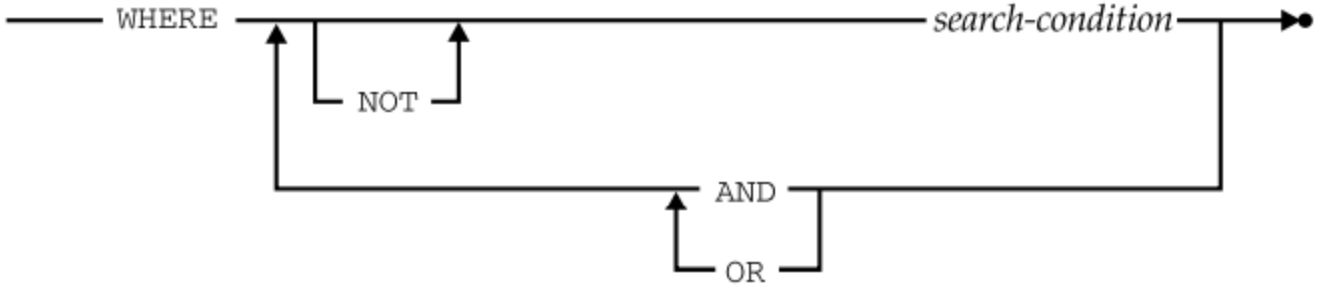
Null Value Test

Search Conditions in Where Clause



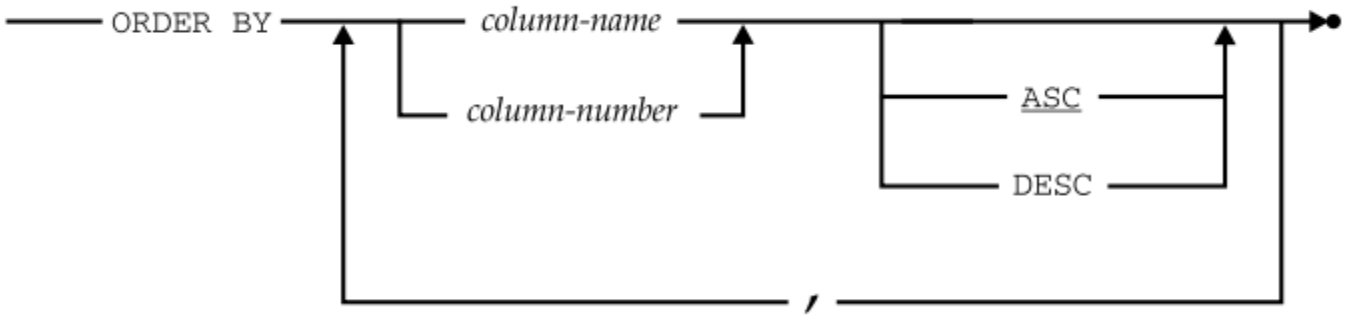
Compound Statements

in Where Clause



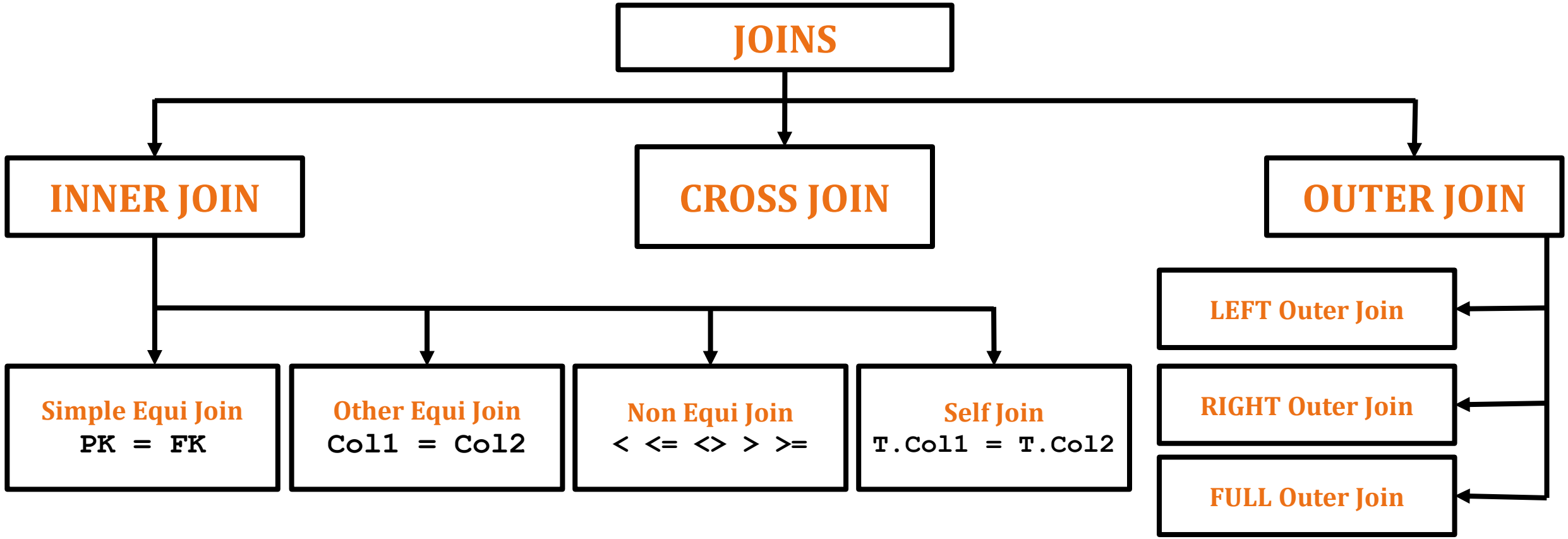
Order by Clause

in Relational Databases



Questions

Joins


JOINS
INNER JOIN
CROSS JOIN
OUTER JOIN
Simple Equi Join
PK = FK
Other Equi Join
Col1 = Col2
Non Equi Join
< <= <> > >=
Self Join
T.Col1 = T.Col2
LEFT Outer Join
RIGHT Outer Join
FULL Outer Join

Let's see how Join works

pk

CustomerId	Name	Age	Address	ContactNo
C0001	Nick	33	New York	+1888888
C0002	Jenny	29	London	+2066666

Customer

pk fk

AccountNo	AccountType	Cust	Balance
SA001	SA	*C0001*	10,000.00
SA002	SA	*C0002*	23,000.00
FD001	FD	*C0001*	2,50,000.00
CA001	CA	*C0001*	75,000.00
FD002	FD	*C0002*	3,50,000.00

Account

Cross Join

```
SELECT  *
FROM Customers, Account;
```

```
SELECT  *
FROM Customers
CROSS JOIN Account;
```

CustomerId	Name	Age	Address	ContactNo	AccountNo	AccountType	Cust	Balance
C0001	Nick	33	New York	+1888888	SA001	SA	*C0001*	10,000.00
C0001	Nick	33	New York	+1888888	SA002	SA	*C0002*	23,000.00
C0001	Nick	33	New York	+1888888	FD001	FD	*C0001*	2,50,000.00
C0001	Nick	33	New York	+1888888	CA001	CA	*C0001*	75,000.00
C0001	Nick	33	New York	+1888888	FD002	FD	*C0002*	3,50,000.00
C0002	Jenny	29	London	+2066666	SA001	SA	*C0001*	10,000.00
C0002	Jenny	29	London	+2066666	SA002	SA	*C0002*	23,000.00
C0002	Jenny	29	London	+2066666	FD001	FD	*C0001*	2,50,000.00
C0002	Jenny	29	London	+2066666	CA001	CA	*C0001*	75,000.00
C0002	Jenny	29	London	+2066666	FD002	FD	*C0002*	3,50,000.00

Equi Join

Step 1

```
SELECT  Name, Address, AccountNo, AccountType, Balance
FROM Customers, Account
WHERE CustomerId = Cust;
```

CustomerId	Name	Age	Address	ContactNo	AccountNo	AccountType	Cust	Balance
C0001	Nick	33	New York	+1888888	SA001	SA	C0001	10,000.00
C0001	Nick	33	New York	+1888888	SA002	SA	C0002	23,000.00
C0001	Nick	33	New York	+1888888	FD001	FD	C0001	2,50,000.00
C0001	Nick	33	New York	+1888888	CA001	CA	C0001	75,000.00
C0001	Nick	33	New York	+1888888	FD002	FD	C0002	3,50,000.00
C0002	Jenny	29	London	+2066666	SA001	SA	C0001	10,000.00
C0002	Jenny	29	London	+2066666	SA002	SA	C0002	23,000.00
C0002	Jenny	29	London	+2066666	FD001	FD	C0001	2,50,000.00
C0002	Jenny	29	London	+2066666	CA001	CA	C0001	75,000.00
C0002	Jenny	29	London	+2066666	FD002	FD	C0002	3,50,000.00

Equi Join

Step 2

```
SELECT  Name, Address, AccountNo, AccountType, Balance
FROM Customers, Account
WHERE CustomerId = Cust;
```

CustomerId	Name	Age	Address	ContactNo	AccountNo	AccountType	Cust	Balance
C0001	Nick	33	New York	+1888888	SA001	SA	C0001	10,000.00
C0001	Nick	33	New York	+1888888	FD001	FD	C0001	2,50,000.00
C0001	Nick	33	New York	+1888888	CA001	CA	C0001	75,000.00
C0002	Jenny	29	London	+2066666	SA002	SA	C0002	23,000.00
C0002	Jenny	29	London	+2066666	FD002	FD	C0002	3,50,000.00

Equi Join

Step 3

```
SELECT  Name, Address, AccountNo, AccountType, Balance
FROM Customers, Account
WHERE CustomerId = Cust;
```

Name	Address	AccountNo	AccountType	Balance
Nick	New York	SA001	SA	10,000.00
Nick	New York	FD001	FD	2,50,000.00
Nick	New York	CA001	CA	75,000.00
Jenny	London	SA002	SA	23,000.00
Jenny	London	FD002	FD	3,50,000.00

Self Join

SalesReps → e

EMPL_NUM	NAME	MGR
1	Nick	1
2	Nancy	4
3	Sophie	1
4	Donna	4
5	Kate	1

SalesReps → m

EMPL_NUM	NAME	MGR
1	Nick	1
2	Nancy	4
3	Sophie	1
4	Donna	4
5	Kate	1

```
SELECT   e.NAME as 'Employee',
         m.NAME as 'Manager'
FROM     SalesReps e, SalesReps m
WHERE    e.MGR = m.EMPL_NUM;
```

Output

Employee	Manager
Nick	Nick
Nancy	Donna
Sophie	Nick
Donna	Donna
Kate	Nick

Let's see how Outer Join works

Globetrotter

Name	Wish
Nancy	Natural Beauty
Cersei	GOT Destinations
Dakota	Wild Life
Sandra	Natural Beauty
Jenny	Beaches
Sophie	NULL

Destination

Country	Attraction
New Zealand	Natural Beauty
Italy	Ancient Structures
Maldives	Beaches
South Africa	Wild Life
Hawaiian Islands	Beaches
England	NULL

INNER Join

```
SELECT *
FROM Globetrotter, Destination
WHERE Wish = Attraction;

SELECT *
FROM Globetrotter INNER JOIN Destination
ON Wish = Attraction;
```

Name	Wish	Country	Attraction
Nancy	Natural Beauty	New Zeland	Natural Beauty
Dakota	Wild Life	South Africa	Wild Life
Sandra	Natural Beauty	New Zeland	Natural Beauty
Jenny	Beaches	Maldives	Beaches
Jenny	Beaches	Hawaiian Islands	Beaches

LEFT Outer Join

```
SELECT *
FROM Globetrotter
LEFT OUTER JOIN Destination
ON Wish = Attraction;
```

Name	Wish	Country	Attraction
Nancy	Natural Beauty	New Zeland	Natural Beauty
Dakota	Wild Life	South Africa	Wild Life
Sandra	Natural Beauty	New Zeland	Natural Beauty
Jenny	Beaches	Maldives	Beaches
Jenny	Beaches	Hawaiian Islands	Beaches
Cersei	GOT Destinations	NULL	NULL
Sophie	NULL	NULL	NULL

RIGHT Outer Join

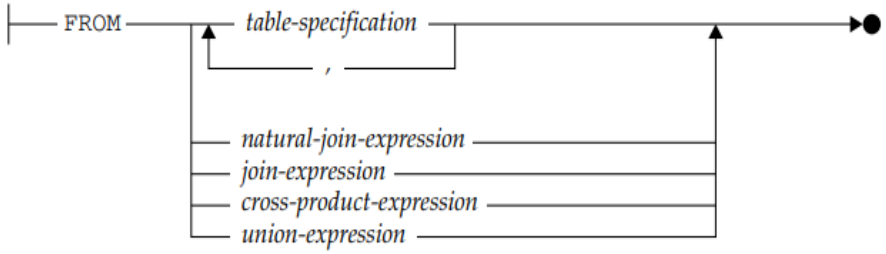
```
SELECT *
FROM Globetrotter
RIGHT OUTER JOIN Destination
ON Wish = Attraction;
```

Name	Wish	Country	Attraction
Nancy	Natural Beauty	New Zeland	Natural Beauty
Dakota	Wild Life	South Africa	Wild Life
Sandra	Natural Beauty	New Zeland	Natural Beauty
Jenny	Beaches	Maldives	Beaches
Jenny	Beaches	Hawaiian Islands	Beaches
NULL	NULL	England	NULL
NULL	NULL	Italy	Ancient Structures

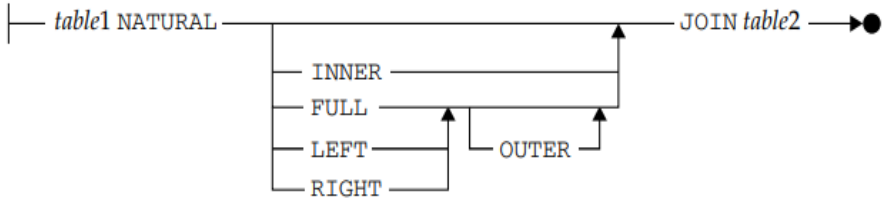
FULL Outer Join

```
SELECT *
FROM Globetrotter
FULL OUTER JOIN Destination
ON Wish = Attraction;
```

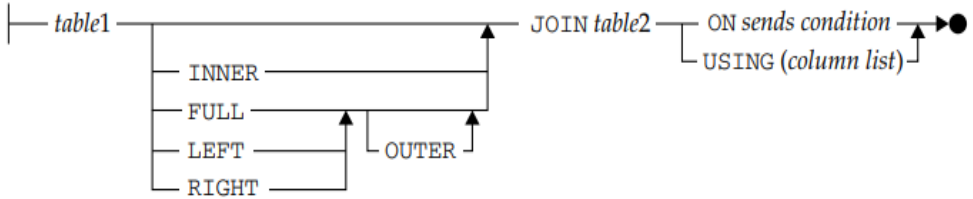
Name	Wish	Country	Attraction
Nancy	Natural Beauty	New Zeland	Natural Beauty
Dakota	Wild Life	South Africa	Wild Life
Sandra	Natural Beauty	New Zeland	Natural Beauty
Jenny	Beaches	Maldives	Beaches
Jenny	Beaches	Hawaiian Islands	Beaches
Cersei	GOT Destinations	NULL	NULL
Sophie	NULL	NULL	NULL
NULL	NULL	England	NULL
NULL	NULL	Italy	Ancient Structures



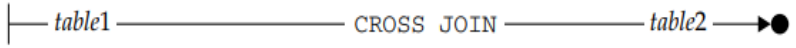
natural-join expression:



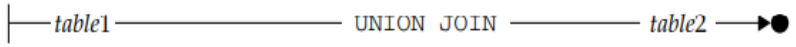
join expression:



cross-product expression:



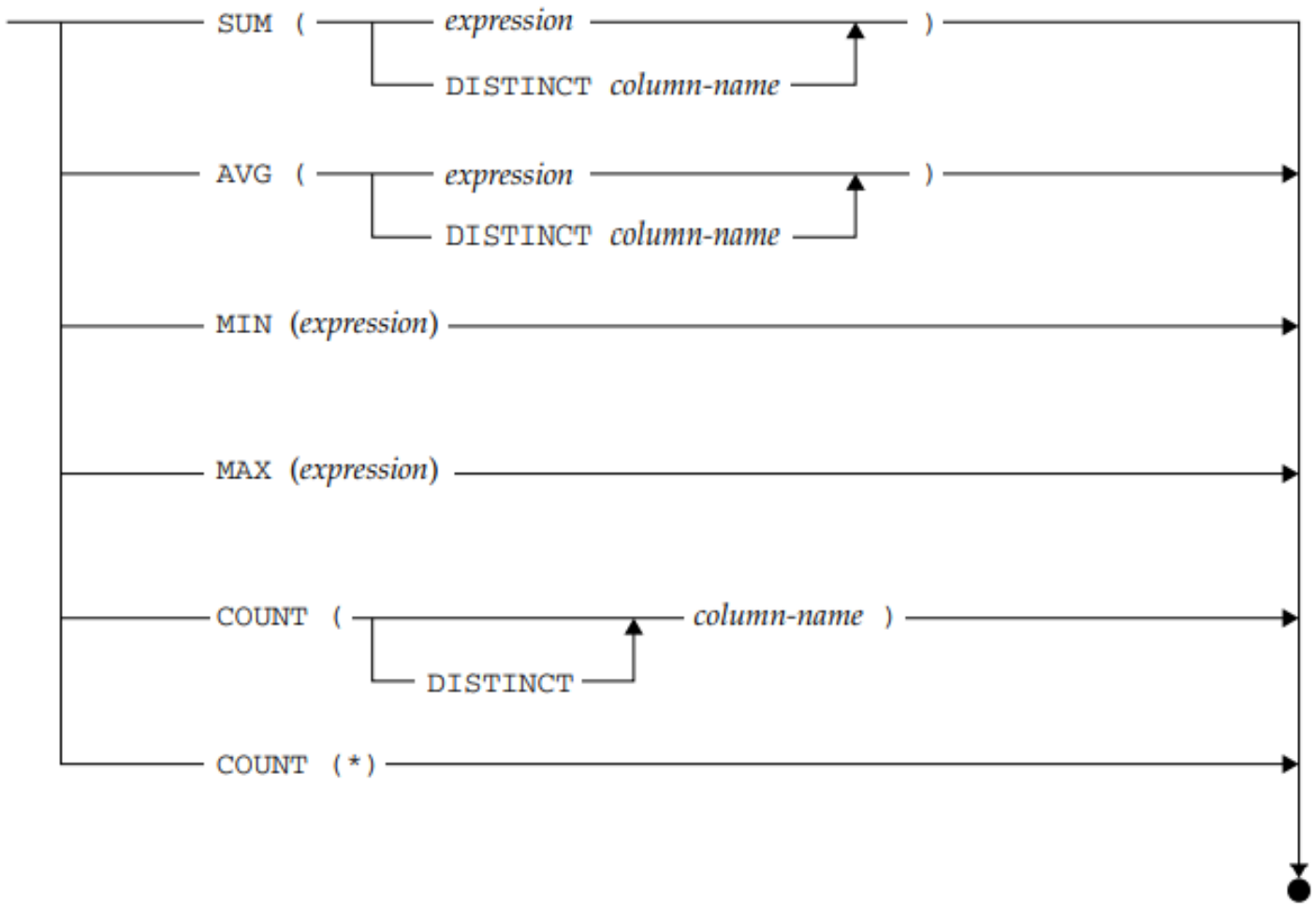
union-expression:



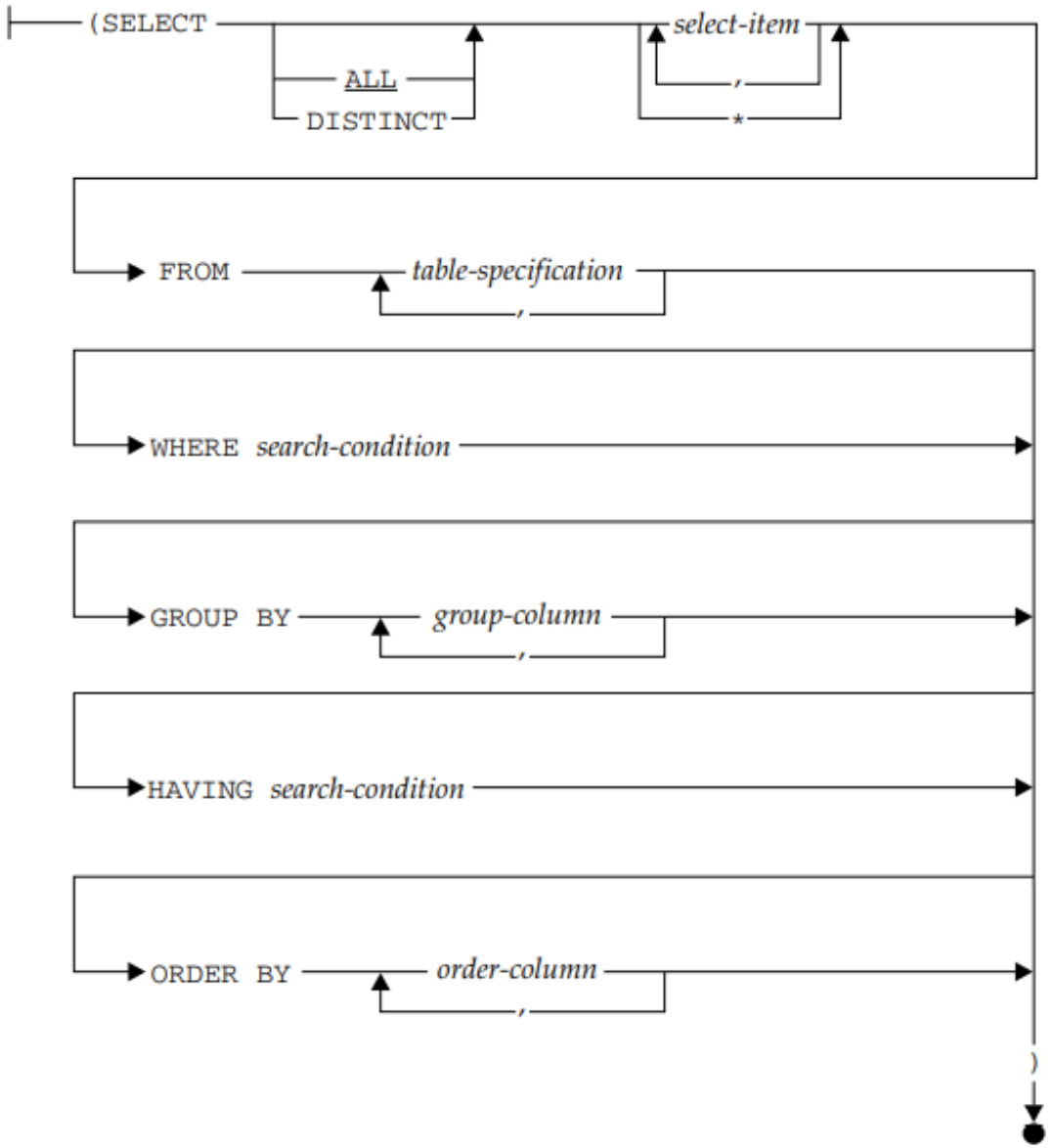
From Clause

in depth

Questions



Column Functions



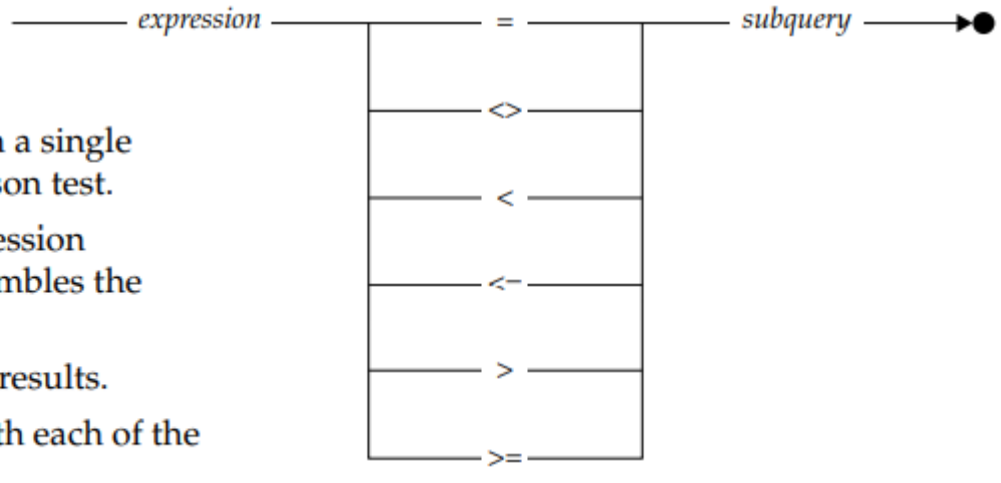
SQL Commands in Relational Databases

Questions

Subquery

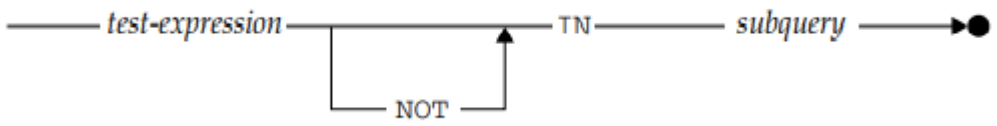
in Relational Databases

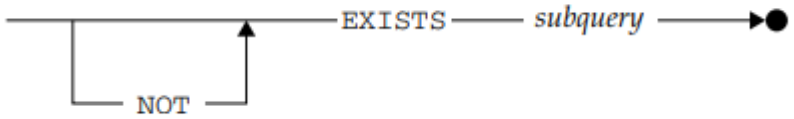
- **Subquery comparison test** Compares the value of an expression with a single value produced by a subquery. This test resembles the simple comparison test.
- **Subquery set membership test** Checks whether the value of an expression matches one of the set of values produced by a subquery. This test resembles the simple set membership test.
- **Existence test** Tests whether a subquery produces any rows of query results.
- **Quantified comparison test** Compares the value of an expression with each of the sets of values produced by a subquery.



Subquery

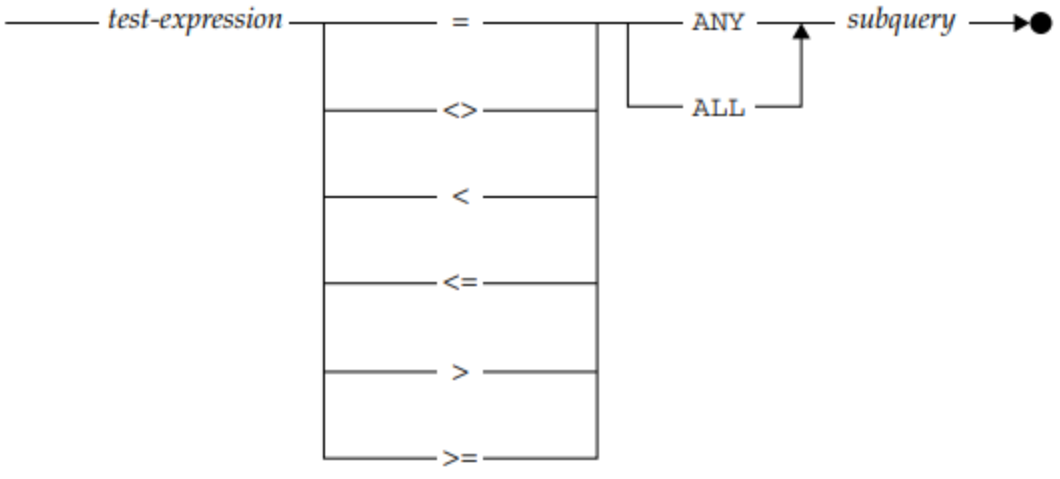
set membership test





Subquery

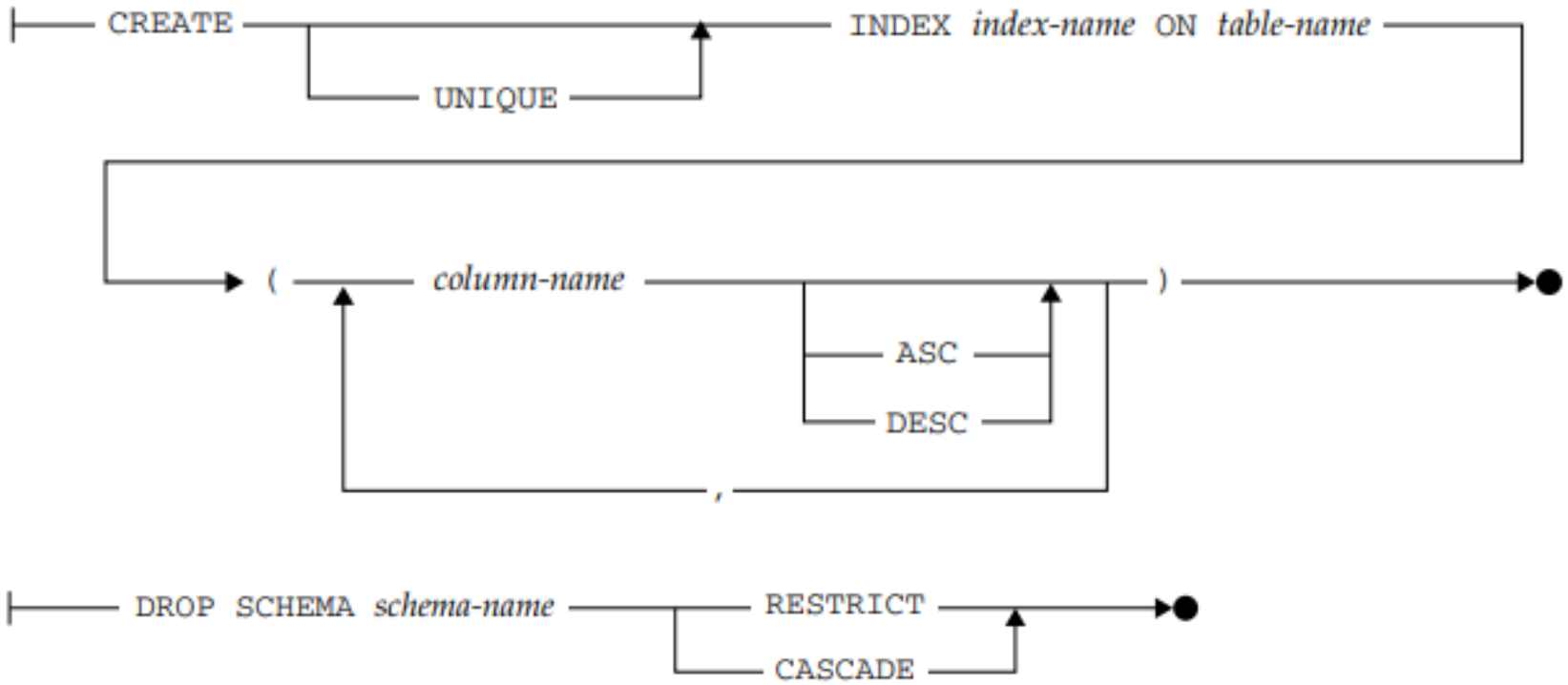
comparison test



CREATE TABLE

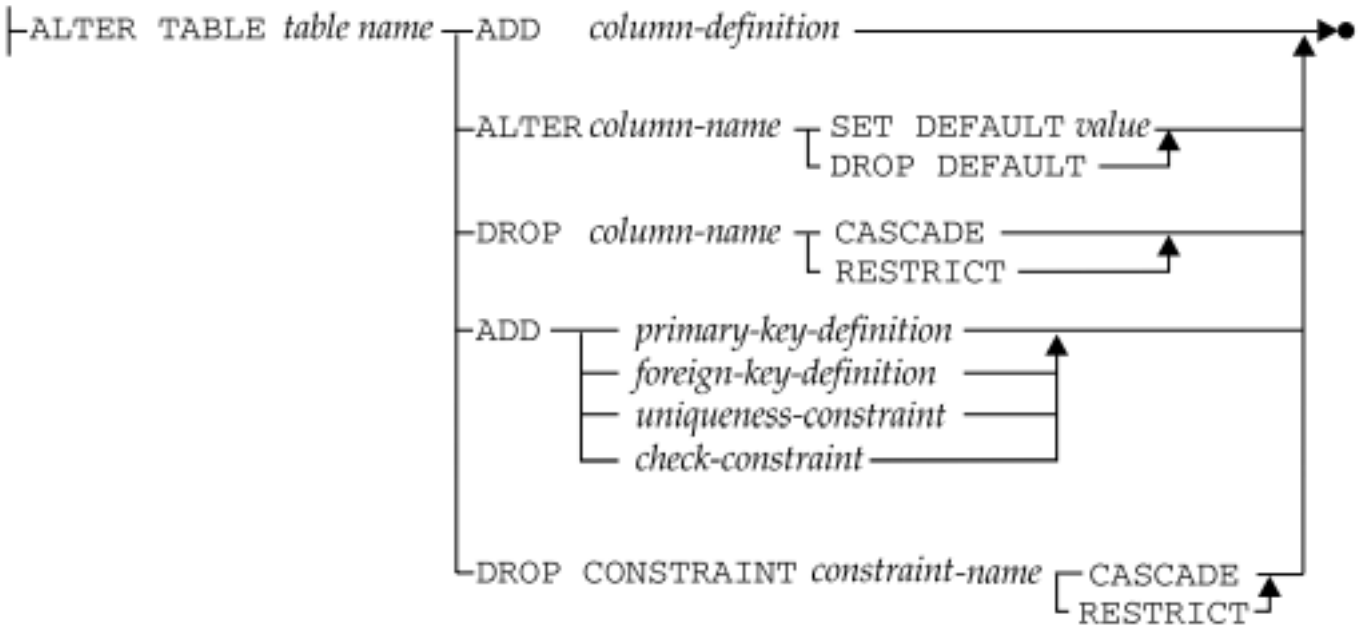
INDEX

in Relational Databases



ALTER

in Relational Databases



TRUNCATE

in Relational Databases

TRUNCATE TABLE statement is a DDL command so it can not be rolled back.

The Truncate command resets the AUTO_INCREMENT counters on the table.

MySQL truncates the table by dropping and creating the table. Thus, the DELETE triggers for the table do not fire during the truncation.

Truncate statement is equivalent to DELETE operation without a WHERE clause.

The truncate command removes the records from a table without scanning it, therefore it is faster than the DELETE statement.

DROP

in Relational Databases

DROP statement is a Data Definition Language(DDL) Command which is used to delete existing database objects. It can be used to delete databases, tables, views, triggers, etc.

A DROP statement in SQL removes a component from a relational database management system (RDBMS).

DROP is a DDL Command. Objects deleted using DROP are permanently lost and it cannot be rolled back. Unlike TRUNCATE which only deletes the data of the tables, the DROP command deletes the data of the table as well as removes the entire schema/structure of the table from the database.
DROP Command removes the table definition and all the data, indexes, triggers, constraints and permission specifications for that table.

Data Integrity

in Relational Databases

1) Required data Some columns in a database must contain a valid data value in every row; they are not allowed to contain missing or `NULL` values.

NOT NULL

2) Validity checking Every column in a database has a *domain,* a set of data values that are legal for that column.

DOMAINS & CHECK

3) Entity integrity The primary key of a table must contain a unique value in each row, which is different from the values in all other rows.

PRIMARY KEY & UNIQUE

4) Referential integrity A foreign key in a relational database links each row in the child table containing the foreign key to the row of the parent table containing the matching primary key value.

FOREIGN KEY

Data Integrity

in Relational Databases

5) Other data relationships The real-world situation modeled by a database will often have additional constraints that govern the legal data values that may appear in the database.
ASSERTION

6) Business rules Updates to a database may be constrained by business rules governing the real-world transactions that are represented by the updates.
TRIGGER

7) Consistency Many real-world transactions cause multiple updates to a database. The DBMS can be asked to enforce this type of consistency rule or to support applications that implement such rules.
TRIGGER

Required Data, Entity Integrity & Referential Integrity

```
CREATE TABLE OFFICES
(
      Office INT PRIMARY KEY,
      City VARCHAR(60) UNIQUE,
      Region VARCHAR(60) NOT NULL,
      Manager INT,
      Sales NUMERIC(10,2),
      Target NUMERIC(10,2),
      FOREIGN KEY (Manager) REFERENCES Salesreps(Empl_num)
)
```

Delete and Update anamolies

Delete

1) **Restrict**
2) **Cascade**
3) **Set Null**
4) **Set Default**

Update

1) **Restrict**
2) **Cascade**
3) **Set Null**
4) **Set Default**

pk

Offices

Office	City	Region	.....
10	London	*Northern*	
20	New York	*Western*	
30	Dubai	*Central*	

pk *fk*

Empl_Num	Name	Rep_Office	.....
101	Nick	*10*	
102	Donna	*10*	
103	Jennifer	*20*	
104	Selin	*10*	
105	Kate	*20*	

Salesreps

Check & Domain

```
CREATE DOMAIN Valid_employee_id INTEGER
CHECK (VALUE BETWEEN 101 AND 199)

CREATE DOMAIN Valid_age INTEGER
CHECK (VALUE >= 18)

CREATE DOMAIN Valid_gender CHAR(1)
CHECK (VALUE IN ('M','F'))

CREATE TABLE Salesreps (
        Empl_Num Valid_employee_id PRIMARY KEY,
        Name VARCHAR(60) NOT NULL,
        Age Valid_age,
        Gender Valid_gender,
        Sales NUMERIC(10,2) CHECK (Sales > = 0),
        Quota NUMERIC(10,2) CHECK (Quota > = 0))
```

Assertion

Ensure that an office's target does not exceed the sum of the quotas for its salespeople.

```
CREATE ASSERTION Valid_target
      CHECK ((Offices.Target <= SUM(Salesreps.Quota) AND
              (Salesreps.Rep_office = Offices.Office))
```

Ensure that the total of the orders for any customer does not exceed their credit limit.

```
CREATE ASSERTION Credit_orders
      CHECK (Customer.Credit_limit <=
             (SELECT SUM(Orders.Amount)
              FROM Orders
              WHERE Orders.Cust = Customers.Cust_Num))
```

Trigger

```
CREATE TRIGGER New_Order
       ON Orders
       FOR INSERT
       AS
              UPDATE Salesreps
              SET Sales = Sales + INSERTED.Amount
              FROM Salesreps, INSERTED
              WHERE Salesreps.Empl_num = INSERTED.Rep

              UPDATE Products
              SET Qty_on_hand = Qty_on_hand - INSERTED.Qty
              FROM Products, INSERTED
              WHERE Products.Mfr_id = INSERTED.Mfr AND
                     Products.Product_id = INSERTED.Product
```

Locks

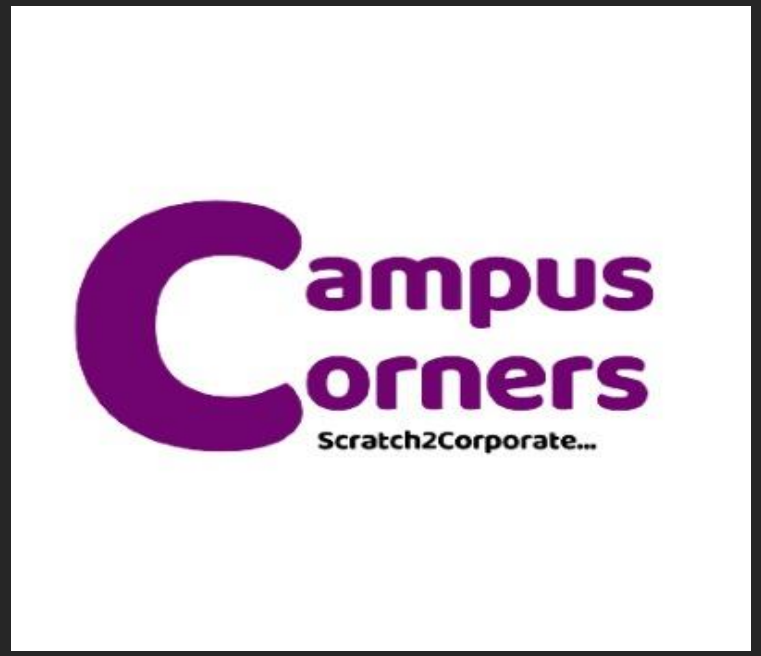
in Relational Databases

Shared and Exclusive Locks

To increase concurrent access to a database, most commercial DBMS products use a locking scheme with more than one type of lock. A scheme using shared and exclusive locks is quite common:

- **Shared lock** Used by the DBMS when a transaction wants to read data from the database. Another concurrent transaction can also acquire a shared lock on the same data, allowing the other transaction to also read the data.
- **Exclusive lock** Used by the DBMS when a transaction wants to update data in the database. When a transaction has an exclusive lock on some data, other transactions cannot acquire any type of lock (shared or exclusive) on the data.

		Transaction B		
		Unlocked	Shared lock	Exclusive lock
Transaction A	Unlocked	OK	OK	OK
	Shared lock	OK	OK	NO
	Exclusive lock	OK	NO	NO



Thank You