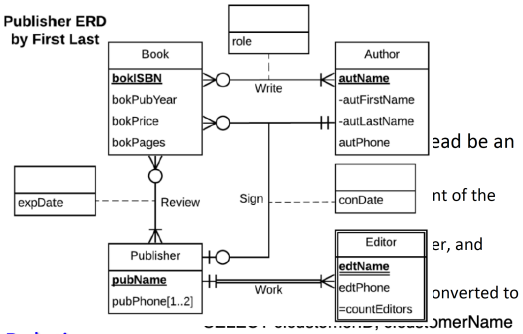


ead be an

nt of the

er, and

onverted to

omerName

Relation

```
FROM Customer_T c
WHERE c.customerID IN
   (SELECT customerID FROM Order_T)
```

- A relation is a named, two-dimensional table of data.
- A table consists of rows (records) and columns (attributes or fields).
- Requirements for a table to qualify as a relation:
  - It must have a unique name.
  - Every attribute value must be atomic (not multivalued, not composite).
  - Every row must be unique.
  - Attributes (columns) in tables must have unique names.
  - The orders of columns and rows must be irrelevant.
- NOTE: All relations are in 1st Normal form (1NF).

- Data Definition Language (**DDL**):
  - Commands that define a database, including creating, altering, and dropping tables and establishing constraints.
- Data Manipulation Language (**DML**):
  - Commands that maintain and query a database.
- Data Control Language (**DCL**):
  - Commands that control a database, including administering privileges and committing data.

```
CREATE TABLE OrderLine_T
   (OrderID              NUMBER(11,0)    NOT NULL,
    ProductID            INTEGER         NOT NULL,
    OrderedQuantity      NUMBER(11,0),
CONSTRAINT OrderLine_PK PRIMARY KEY (OrderID, ProductID),
CONSTRAINT OrderLine_FK1 FOREIGN KEY (OrderID) REFERENCES Order_T(OrderID),
CONSTRAINT OrderLine_FK2 FOREIGN KEY (ProductID) REFERENCES Product_T(ProductID));
```

```
SELECT ProductDescription, ProductFinish, P
  FROM Product_T;
  WHERE (ProductDescription LIKE '%Desk'
    OR ProductDescription LIKE '%Table')
    AND ProductStandardPrice > 300;
```

```
INSERT INTO CaCustomer_T
SELECT * FROM Customer_T
   WHERE CustomerState = 'CA';
```

```
-- Which city has either an employee or a customer
SELECT e.employeeCity FROM Employee_T e
UNION
SELECT c.CustomerCity FROM Customer_T c

-- Which city has either an employee and a customer
SELECT e.employeeCity FROM Employee_T e
INTERSECT
SELECT c.CustomerCity FROM Customer_T c

-- Which city has an employee but no customer
SELECT e.employeeCity FROM Employee_T e
EXCEPT
SELECT c.CustomerCity FROM Customer_T c
```

- **ALTER TABLE** statement allows you to change column specifications:

```
ALTER TABLE table_name alter_table_action;
```

- Table Actions:

```
ADD [COLUMN] column_definition
ALTER [COLUMN] column_name SET DEFAULT default-value
ALTER [COLUMN] column_name DROP DEFAULT
DROP [COLUMN] column_name [RESTRICT] [CASCADE]
ADD table_constraint
```

- Example (adding a new column with a default value):

```
ALTER TABLE CUSTOMER_T
ADD COLUMN CustomerType VARCHAR2 (2) DEFAULT "Commercial";
```

CUSTOMER entity type with composite attribute
Figure 4-9: Customer (**customerID**, customerName, customerStreet, customerCity, customerState, customerPostalCode)

EMPLOYEE entity type with multivalued attribute
Figure 4-10: Employee (**employeeID**, employeeName, employeeAddress)
Employee_Skill (***employeeID***, **skill**)

Weak entity DEPENDENT with composite name pk
Figure 4-11: Employee (**employeeID**, employeeName)
Dependent (***employeeID***, **dependentFirstName**, **dependentMiddleInitial**, **dependentLastName**, dependentDateOfBirth, dependentGender)

Add foreign key to the M optional entity type 1:M
Figure 4-12: Customer (**customerID**, customerName, customerAddress, customerPostalCode)
Order (**orderID**, orderDate, *customerID*)

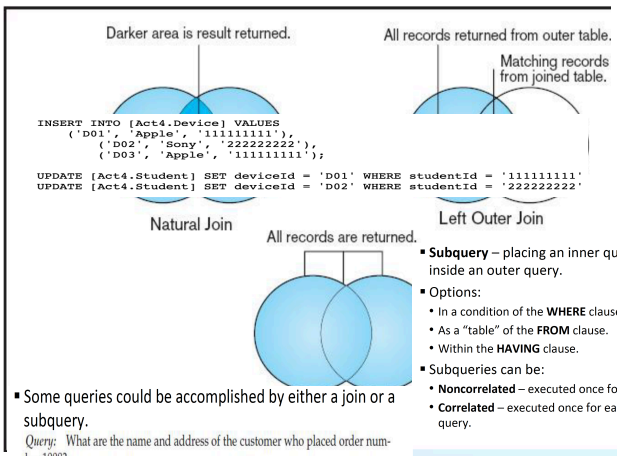
Create new relation with two foreign keys and attribute M:M
Figure 4-13: Employee (**employeeID**, employeeName, employeeDateOfBirth)
Course (**courseID**, courseTitle)
Complete (***employeeID***, ***courseID***, dateCompleted)

Add foreign key and attribute to the optional entity type
Figure 4-14: Nurse (**NurseID**, nurseName, nurseBirthName)
CareCenter (**centerID**, centerLocation, *nurseID*, dateAssigned)

Two strong entities and an associative entity (no own pk)
Figure 4-15: Order (**orderID**, orderDate)
Product (**productID**, productDescription, productFinish, productStandardPrice, productLineID)
OrderLine (***orderID***, ***productID***, orderedQuantity)

Two strong entities and an associative entity (have own pk)
Figure 4-16: Customer (**customerID**, customerName)
Vendor (**vendorID**, vendorAddress)
Shipment (*customerID*, *vendorID*, **shipmentID**, shipmentDate, shipmentAmount)

Unary 1:N Relationship (M:N need new relation)
Figure 4-17: Employee (**employeeID**, employeeName, employeeDateOfBirth, *managerEmployeeID*)



- **Subquery** – placing an inner query (**SELECT** statement) inside an outer query.
- Options:
  - In a condition of the **WHERE** clause.
  - As a "table" of the **FROM** clause.
  - Within the **HAVING** clause.
- Subqueries can be:
  - **Noncorrelated** – executed once for the entire outer query.
  - **Correlated** – executed once for each row returned by the outer query.

- Some queries could be accomplished by either a join or a subquery.

Query: What are the name and address of the customer who placed order number 1008?

Join version

```
SELECT CustomerName, CustomerAddress, CustomerCity,
  CustomerState, CustomerPostalCode
FROM Customer_T, Order_T
WHERE Customer_T.CustomerID = Order_T. CustomerID
  AND OrderID = 1008;
```

Subquery version

```
SELECT CustomerName, CustomerAddress, CustomerCity,
CustomerState, CustomerPostalCode
  FROM Customer_T
    WHERE Customer_T.CustomerID =
      (SELECT Order_T.CustomerID
        FROM Order_T
          WHERE OrderID = 1008);
```

```
{CASE expression
{WHEN expression
THEN {expression | NULL}} . . .
| {WHEN predicate
THEN {expression | NULL}} . . .
[ELSE {expression  NULL}]
END }
| { NULLIF (expression, expression) }
| { COALESCE (expression . . .) }
```

- Show all products whose standard price is higher than the average price.

Subquery forms the derived table used in the FROM clause of the outer query

One column of the subquery is an aggregate function that has an alias name. That alias can then be referred to in the outer query.

```
SELECT ProductDescription, ProductStandardPrice, AvgPrice
  FROM
    (SELECT AVG(ProductStandardPrice) AvgPrice FROM Product_T),
      Product_T
  WHERE ProductStandardPrice > AvgPrice;
```

The WHERE clause normally cannot include aggregate functions, but because the aggregate is performed in the subquery its result can be used in the outer query's WHERE clause.

What are the names of all customers, who have placed orders? A statement using **subquery** here:

Your Answer:

```
SELECT customerName FROM Customer_T
WHERE customerID IN (
    SELECT customerID
    FROM Order_T
    )
```

- **Data Warehouse** – A subject-oriented, integrated, time-variant, non-updatable collection of data used in support of management decision-making processes.
  - **Subject-oriented**: e.g. customers, patients, students, products.
  - **Integrated**: consistent naming conventions, formats, encoding structures; from multiple data sources.
  - **Time-variant**: can study trends and changes.
  - **Non-updatable**: read-only, periodically refreshed.

- **Data Mart** – A data warehouse that is limited in scope.

TABLE 9-1 Comparison of Operational and Informational Systems

Characteristic	Operational Systems	Informational Systems
Primary purpose	Run the business on a current basis	Support managerial decision making
Type of data	Current representation of state of the business	Historical point-in-time (snapshots) and predictions
Primary users	Clerks, salespersons, administrators	Managers, business analysts, customers
Scope of usage	Narrow, planned, and simple updates and queries	Broad, ad hoc, complex queries and analysis
Design goal	Performance: throughput, availability	Ease of flexible access and use
Volume	Many constant updates and queries on one or a few table rows	Periodic batch updates and queries requiring many or all rows

Data Warehouse Architectures

- Independent data mart.
- Dependent data mart and operational data store.
- Logical data mart and real-time data warehouse.
- Three-layer architecture.

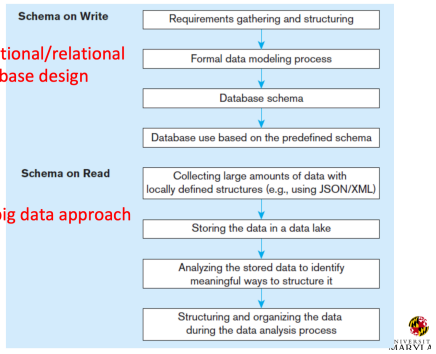
All involve some form of ***extract***, ***transform*** and ***load*** (**ETL**).

Table 9-2: Data Warehouse Versus Data Mar

TABLE 9-2 Data Warehouse Versus Data Mart

Data Warehouse	Data Mart
Scope	**Scope**
• Application independent • Centralized, possibly enterprise-wide • Planned	• Specific DSS application • Decentralized by user area • Organic, possibly not planned
Data	**Data**
• Historical, detailed, and summarized • Lightly denormalized	• Some history, detailed, and summarized • Highly denormalized
Subjects	**Subjects**
• Multiple subjects	• One central subject of concern to users
Sources	**Sources**
• Many internal and external sources	• Few internal and external sources
Other Characteristics	**Other Characteristics**
• Flexible • Data oriented • Long life • Large • Single complex structure	• Restrictive • Project oriented • Short life • Start small, becomes large • Multi, semi-complex structures, together complex

Figure 11-2: Schema on Write versus Schema on Read



traditional/relational database design

big data approach

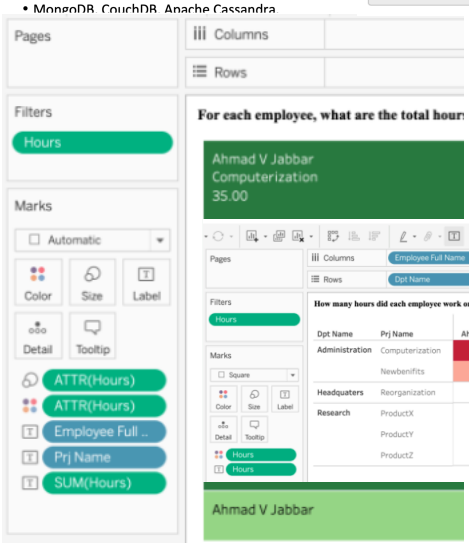
Other Data Warehouse Advances

- Move data warehouse into the cloud to enjoy the benefits of lower total cost of ownership (TCO).
  - IBM, Oracle, Microsoft, Teradata, SAP (HANA), Amazon (Redshift).
- Columnar databases:
  - Issue of Big Data (huge volume, often unstructured).
  - Optimize storage for summary data of few columns.
  - Sybase, Vertica, Infobright.
- NoSQL:
  - "Not only SQL".
  - Deals with unstructured data.
  - MongoDB, CouchDB, Apache Cassandra

```
CREATE VIEW rstAvgRating AS
SELECT r.rstId, CAST(AVG (rv.rvwRating) AS DECIMAL(3,2)) AS avgRating
FROM [Master_Lee's.Restaurant] r, [Master_Lee's.Review] rv
WHERE r.rstId=rv.rstId
GROUP BY r.rstId
```

What are the five Vs to define big data according to the textbook?

Volume	Variety	Velocity	Veracity
Value			



- **Descriptive analytics** was the original emphasis of BI
- Reporting of aggregate quantitative query results
- Tabular or data visualization displays
- **Dashboard**: a few key indicators
- **Scorecard**: like a dashboard, but broader range
- **OLAP**: online analytical processing

Other Hadoop

- **Pig**
  - A tool that integra environment inter
  - Useful developme
- **Hive**
  - An Apache project large data sets usi declarative interfa
  - Useful for ETL task
- **HBase**

Table 11-2: NoSQL Comparison

TABLE 11-2 Comparison of NoSQL Database Characteristics (Based on Scofield, 2010)

	Key-Value Store	Document Store	Column Oriented	Graph
Performance	high	high	high	variable
Scalability	high	variable/high	high	variable
Flexibility	high	high	moderate	high
Complexity	none	low	low	high
Functionality	variable	variable (low)	minimal	graph theory

Source: http://www.slideshare.net/bscofield/nosql-codemash-2010
Courtesy of Ben Scofield.

- NoSQL Examples:
  - Redis: Key-value store DBMS
  - MongoDB: document store DBMS
  - Apache Cassandra: wide-column store DBMS
  - Neo4j: graph DBMS