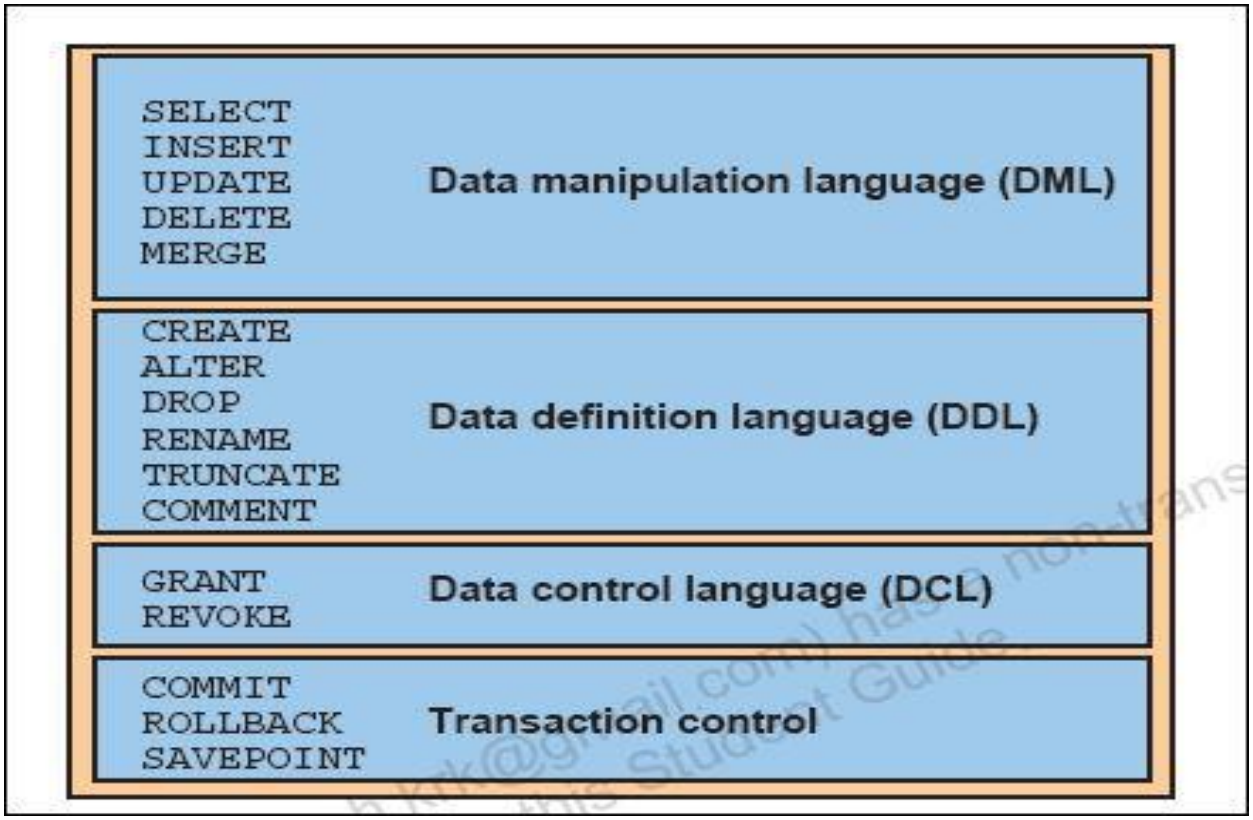
Name Of Subject : Database Management System

Unit-3:- SQL Concepts

SQL Basics

- Structured Query Language (SQL)
- SQL Statements



SQL Basics

- SQL Statements

Statement	Description
SELECT INSERT UPDATE DELETE MERGE	Retrieves data from the database, enters new rows, changes existing rows, and removes unwanted rows from tables in the database, respectively. Collectively known as *data manipulation language* (DML).
CREATE ALTER DROP RENAME TRUNCATE COMMENT	Sets up, changes, and removes data structures from tables. Collectively known as *data definition language* (DDL).
GRANT REVOKE	Gives or removes access rights to both the Oracle database and the structures within it.
COMMIT ROLLBACK SAVEPOINT	Manages the changes made by DML statements. Changes to the data can be grouped together into logical transactions.

Domain Types in SQL

- **char(n).** Fixed length character string, with user-specified length *n.*
- **varchar(n).** Variable length character strings, with user-specified maximum length *n.*
- **int.** Integer (a finite subset of the integers that is machine-dependent
- **numeric(p,d).** Fixed point number, with user-specified precision of *p* digits, with *n* digits to the right of decimal point.
- **float(n).** Floating point number, with user-specified precision of at least *n* digits.

Steps in Table Creation

1. Identify data types for attributes
2. Identify columns that can and cannot be null
3. Identify columns that must be unique (candidate keys)
4. Identify primary key–foreign key mates
5. Determine default values
6. Identify constraints on columns (domain specifications)
7. Create the table and associated indexes

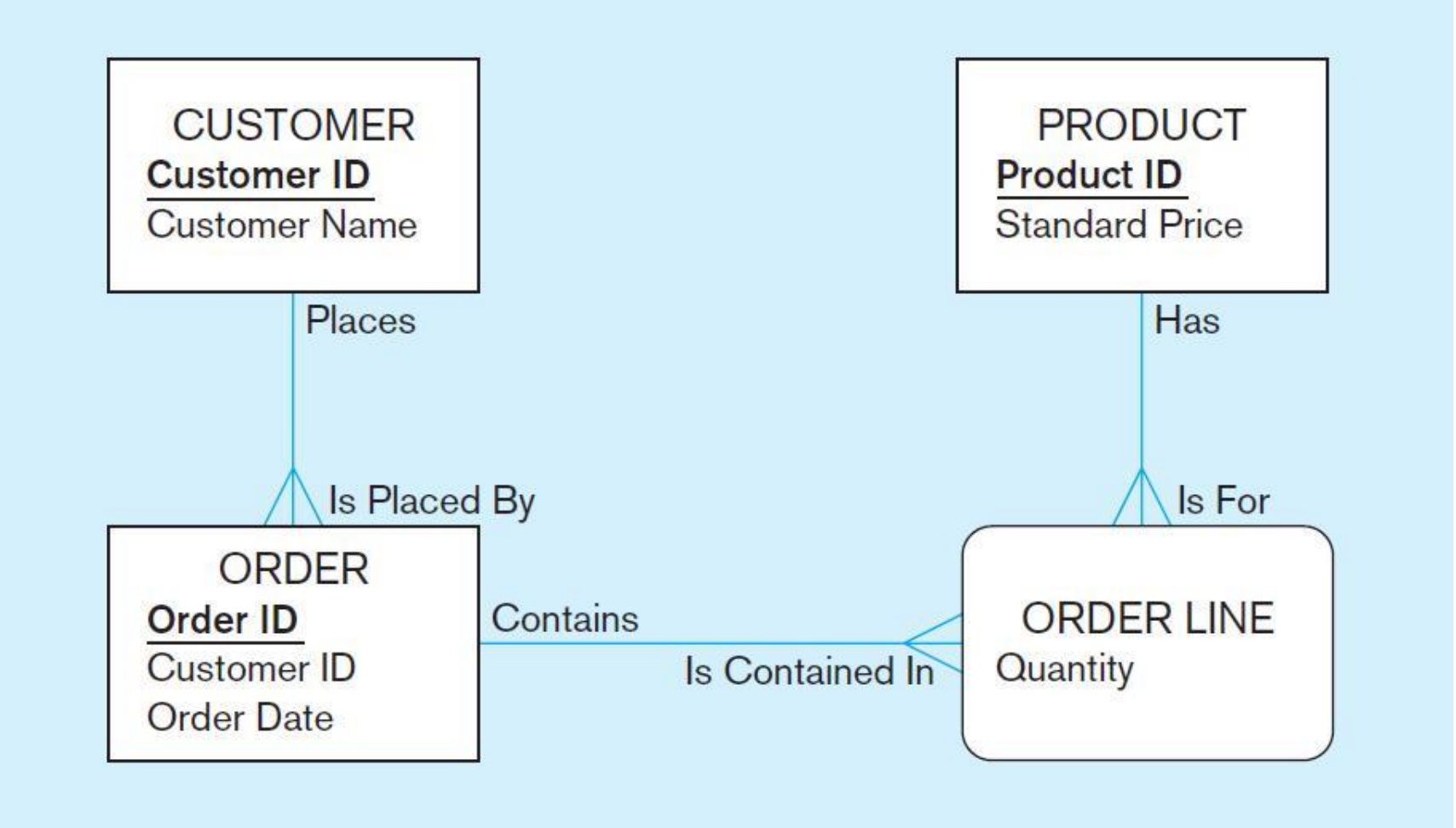
General syntax for CREATE TABLE statement used in data definition language

```
CREATE TABLE tablename
( {column definition   [table constraint] } . , . .
[ON COMMIT {DELETE | PRESERVE} ROWS] );

where column definition ::=
column_name
        {domain name | datatype [(size)] }
        [column_constraint_clause . . .]
        [default value]
        [collate clause]

and table constraint ::=
        [CONSTRAINT constraint_name]
        Constraint_type [constraint_attributes]
```

The following slides create tables for this enterprise data model



SQL database definition commands for Pine Valley Furniture Company (Oracle 11g)

```
CREATE TABLE Customer_T
                (CustomerID                 NUMBER(11,0)      NOT NULL,
                 CustomerName               VARCHAR2(25)      NOT NULL,
                 CustomerAddress            VARCHAR2(30),
                 CustomerCity               VARCHAR2(20),
                 CustomerState              CHAR(2),
                 CustomerPostalCode         VARCHAR2(9),
CONSTRAINT Customer_PK PRIMARY KEY (CustomerID));
```

```
CREATE TABLE Order_T
                (OrderID                    NUMBER(11,0)      NOT NULL,
                 OrderDate                  DATE DEFAULT SYSDATE,
                 CustomerID                 NUMBER(11,0),
CONSTRAINT Order_PK PRIMARY KEY (OrderID),
CONSTRAINT Order_FK FOREIGN KEY (CustomerID) REFERENCES Customer_T(CustomerID));
```

```
CREATE TABLE Product_T
                (ProductID                  NUMBER(11,0)      NOT NULL,
                 ProductDescription         VARCHAR2(50),
                 ProductFinish              VARCHAR2(20)
                        CHECK (ProductFinish IN ('Cherry', 'Natural Ash', 'White Ash',
                                                 'Red Oak', 'Natural Oak', 'Walnut')),
                 ProductStandardPrice       DECIMAL(6,2),
                 ProductLineID              INTEGER,
CONSTRAINT Product_PK PRIMARY KEY (ProductID));
```

```
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));
```

Overall table definitions

Defining attributes and their data types

```
CREATE TABLE Product_T
        (ProductID              NUMBER(11,0)    NOT NULL,
         ProductDescription     VARCHAR2(50),
         ProductFinish          VARCHAR2(20)
                CHECK (ProductFinish IN ('Cherry', 'Natural Ash', 'White Ash',
                       'Red Oak', 'Natural Oak', 'Walnut')),
         ProductStandardPrice   DECIMAL(6,2),
         ProductLineID          INTEGER,
CONSTRAINT Product_PK PRIMARY KEY (ProductID));
```


9

Non-nullable specification

```
CREATE TABLE Product_T
        (ProductID                  NUMBER(11,0)    NOT NULL,
         ProductDescription         VARCHAR2(50),
         ProductFinish              VARCHAR2(20)
                                    CHECK (ProductFinish IN ('Cherry', 'Natural Ash', 'White Ash',
                                           'Red Oak', 'Natural Oak', 'Walnut')),
         ProductStandardPrice       DECIMAL(6,2),
         ProductLineID              INTEGER,
CONSTRAINT Product_PK PRIMARY KEY (ProductID));
```

Primary keys can never have NULL values

Identifying primary key

10

Non-nullable specifications

```
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));
```

Primary key

Some primary keys are composite–composed of multiple attributes

11

Controlling the values in attributes

```
CREATE TABLE Order_T
                (OrderID                    NUMBER(11,0)        NOT NULL,
                 OrderDate                  DATE DEFAULT SYSDATE,
                 CustomerID                 NUMBER(11,0),
CONSTRAINT Order_PK PRIMARY KEY (OrderID),
CONSTRAINT Order_FK FOREIGN KEY (CustomerID) REFERENCES Customer_T(CustomerID));

CREATE TABLE Product_T
                (ProductID                  NUMBER(11,0)        NOT NULL,
                 ProductDescription         VARCHAR2(50),
                 ProductFinish              VARCHAR2(20)
                    CHECK (ProductFinish IN ('Cherry', 'Natural Ash', 'White Ash',
                                             'Red Oak', 'Natural Oak', 'Walnut')),
                 ProductStandardPrice       DECIMAL(6,2),
                 ProductLineID              INTEGER,
CONSTRAINT Product_PK PRIMARY KEY (ProductID));
```

Default value

Domain constraint

12

Identifying foreign keys and establishing relationships

```
CREATE TABLE Customer_T
                (CustomerID                NUMBER(11,0)        NOT NULL,
                 CustomerName              VARCHAR2(25)        NOT NULL,
                 CustomerAddress           VARCHAR2(30),
                 CustomerCity              VARCHAR2(20),
                 CustomerState             CHAR(2),
                 CustomerPostalCode        VARCHAR2(9),
CONSTRAINT Customer_PK PRIMARY KEY (CustomerID));

CREATE TABLE Order_T
                (OrderID                   NUMBER(11,0)        NOT NULL,
                 OrderDate                 DATE DEFAULT SYSDATE,
                 CustomerID                NUMBER(11,0),
CONSTRAINT Order_PK PRIMARY KEY (OrderID),
CONSTRAINT Order_FK FOREIGN KEY (CustomerID) REFERENCES Customer_T(CustomerID));
```

Primary key of parent table

Foreign key of dependent table

13

Data Integrity Controls

- Referential integrity–constraint that ensures that foreign key values of a table must match primary key values of a related table in 1:M relationships
- Restricting:
  - Deletes of primary records
  - Updates of primary records
  - Inserts of dependent records

Changing Tables

- 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";
```

Removing Tables

- DROP TABLE statement allows you to remove tables from your schema:
  - DROP TABLE CUSTOMER_T

Insert Statement

- Adds one or more rows to a table
- Inserting into a table

```
INSERT INTO Customer_T VALUES
(001, 'Contemporary Casuals', '1355 S. Himes Blvd.', 'Gainesville', 'FL', 32601);
```

- Inserting a record that has some null attributes requires identifying the fields that actually get data

```
INSERT INTO Product_T (ProductID,
ProductDescription, ProductFinish, ProductStandardPrice)
    VALUES (1, 'End Table', 'Cherry', 175, 8);
```

- Inserting from another table

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

Creating Tables with Identity Columns

```
CREATE TABLE Customer_T
(CustomerID INTEGER GENERATED ALWAYS AS IDENTITY
     (START WITH 1
     INCREMENT BY 1
     MINVALUE 1
     MAXVALUE 10000
     NO CYCLE),
CustomerName          VARCHAR2(25) NOT NULL,
CustomerAddress       VARCHAR2(30),
CustomerCity          VARCHAR2(20),
CustomerState         CHAR(2),
CustomerPostalCode    VARCHAR2(9),
CONSTRAINT Customer_PK PRIMARY KEY (CustomerID);
```

Introduced with SQL:2008

Inserting into a table does not require explicit customer ID entry or field list

INSERT INTO CUSTOMER_T VALUES ('Contemporary Casuals', '1355 S. Himes Blvd.', 'Gainesville', 'FL', 32601);

Delete Statement

- Removes rows from a table
- Delete certain rows
  - **DELETE FROM CUSTOMER_T WHERE CUSTOMERSTATE = 'HI';**
- Delete all rows
  - **DELETE FROM CUSTOMER_T;**

Update Statement

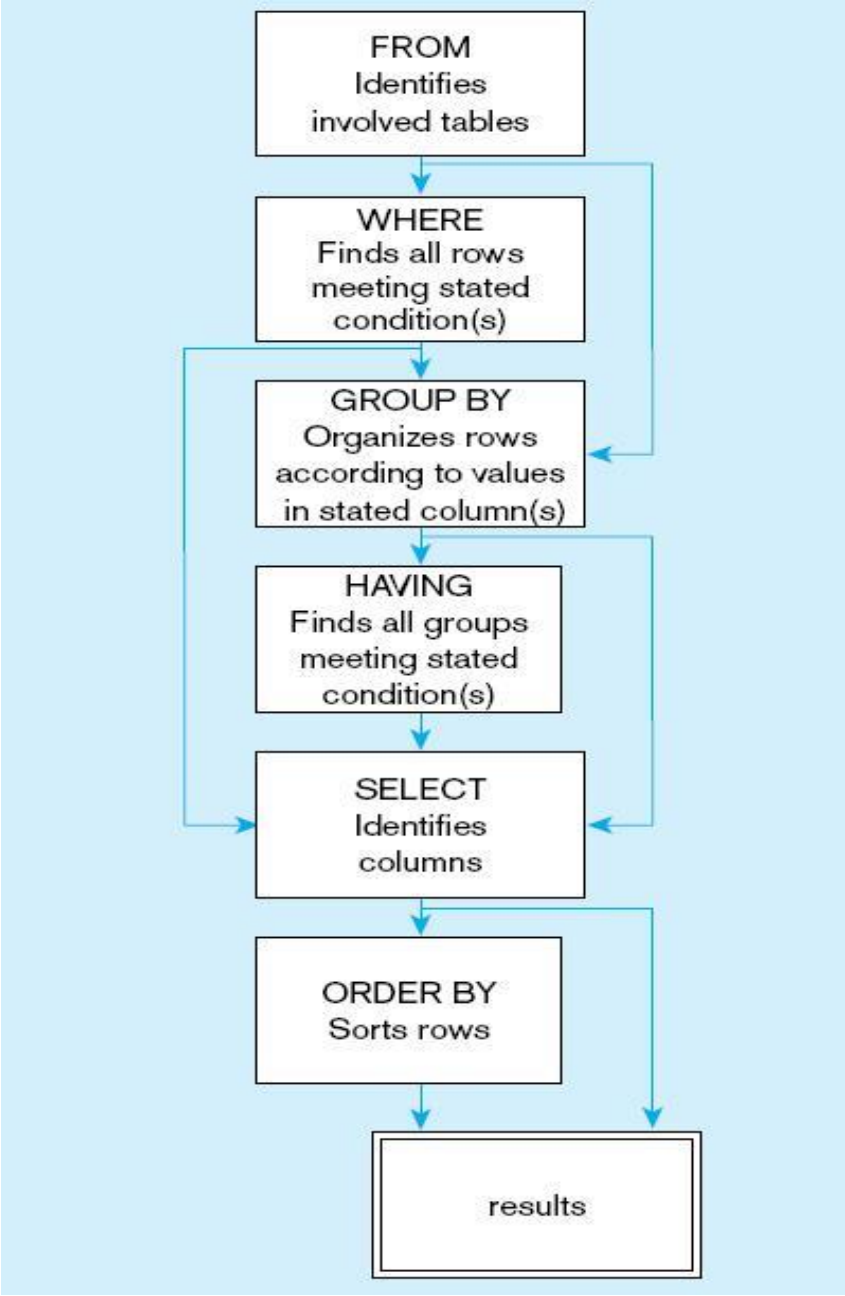
- Modifies data in existing rows

```
UPDATE Product_T
SET ProductStandardPrice = 775
    WHERE ProductID = 7;
```

SELECT Statement

- Used for queries on single or multiple tables
- Clauses of the SELECT statement:
  - **SELECT**
    - List the columns (and expressions) to be returned from the query
  - **FROM**
    - Indicate the table(s) or view(s) from which data will be obtained
  - **WHERE**
    - Indicate the conditions under which a row will be included in the result
  - **GROUP BY**
    - Indicate categorization of results
  - **HAVING**
    - Indicate the conditions under which a category (group) will be included
  - **ORDER BY**
    - Sorts the result according to specified criteria

SQL statement processing order (based on van der Lans, 2006 p.100)



SELECT Example

- Find products with standard price less than $275

```
SELECT ProductDescription, ProductStandardPrice
   FROM Product_T
      WHERE ProductStandardPrice < 275;
```

TABLE 6-3 Comparison Operators in SQL

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to
!=	Not equal to

Table : Comparison Operators in SQL

SELECT Example Using Alias

- Alias is an alternative column or table name

```
SELECT CUST.CustomerName AS Name, CUST.CustomerAddress
   FROM ownerid.Customer_T AS Cust
      WHERE Name = 'Home Furnishings';
```

- Here, CUST is a table alias and Name is a column alias

SELECT Example Using a Function

✖Using the COUNT ***aggregate function*** to find totals

```
SELECT COUNT(*) FROM ORDERLINE_T
    WHERE ORDERID = 1004;
```

Note: with aggregate functions you can't have single-valued columns included in the SELECT clause, unless they are included in the GROUP BY clause.

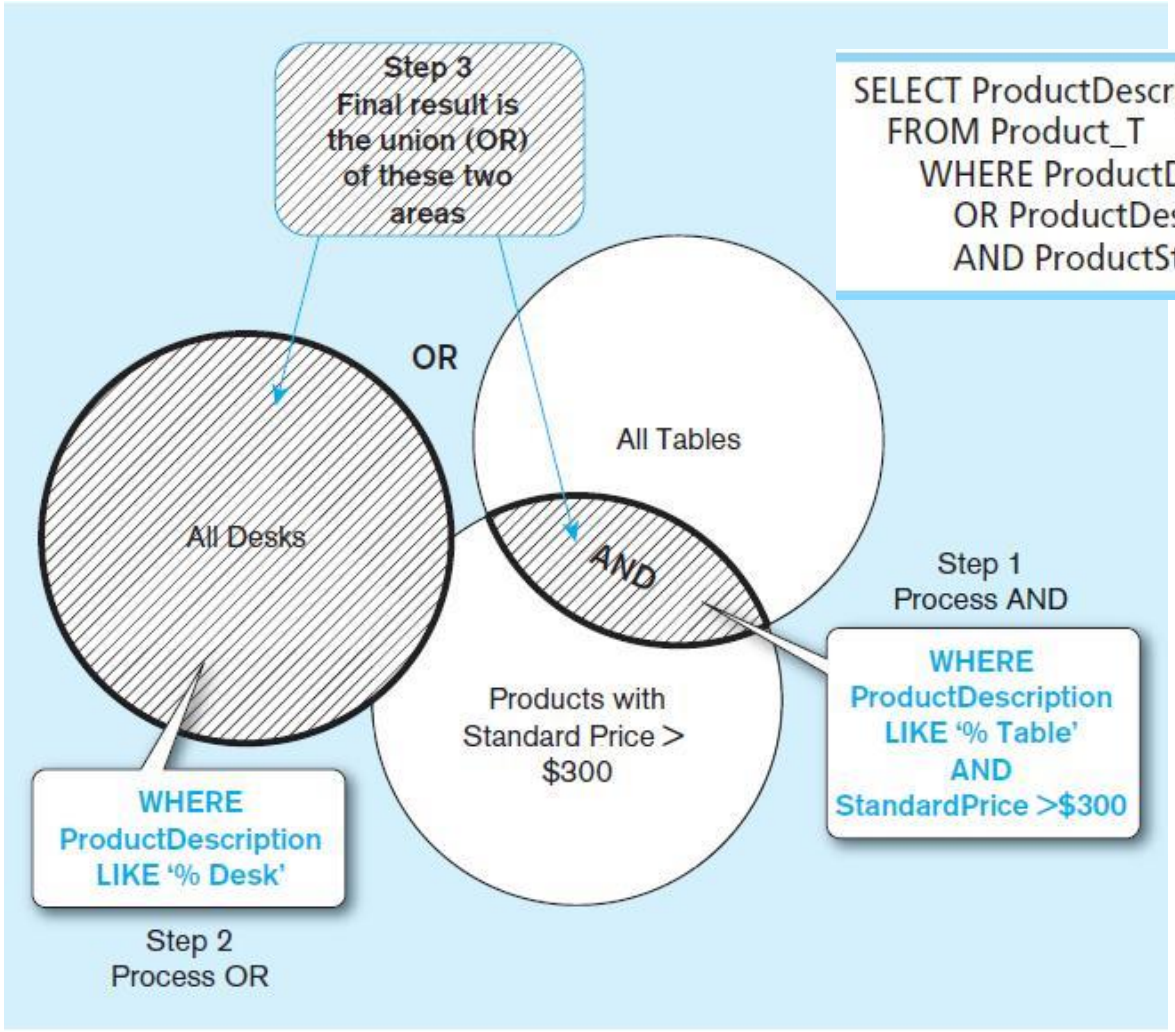
SELECT Example–Boolean Operators

- **AND**, **OR**, and **NOT** Operators for customizing conditions in WHERE clause

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

Note: the **LIKE** operator allows you to compare strings using wildcards. For example, the % wildcard in '%Desk' indicates that all strings that have any number of characters preceding the word "Desk" will be allowed.

Figure 6-7 Boolean query A without use of parentheses




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

By default, processing order of Boolean operators is NOT, then AND, then OR

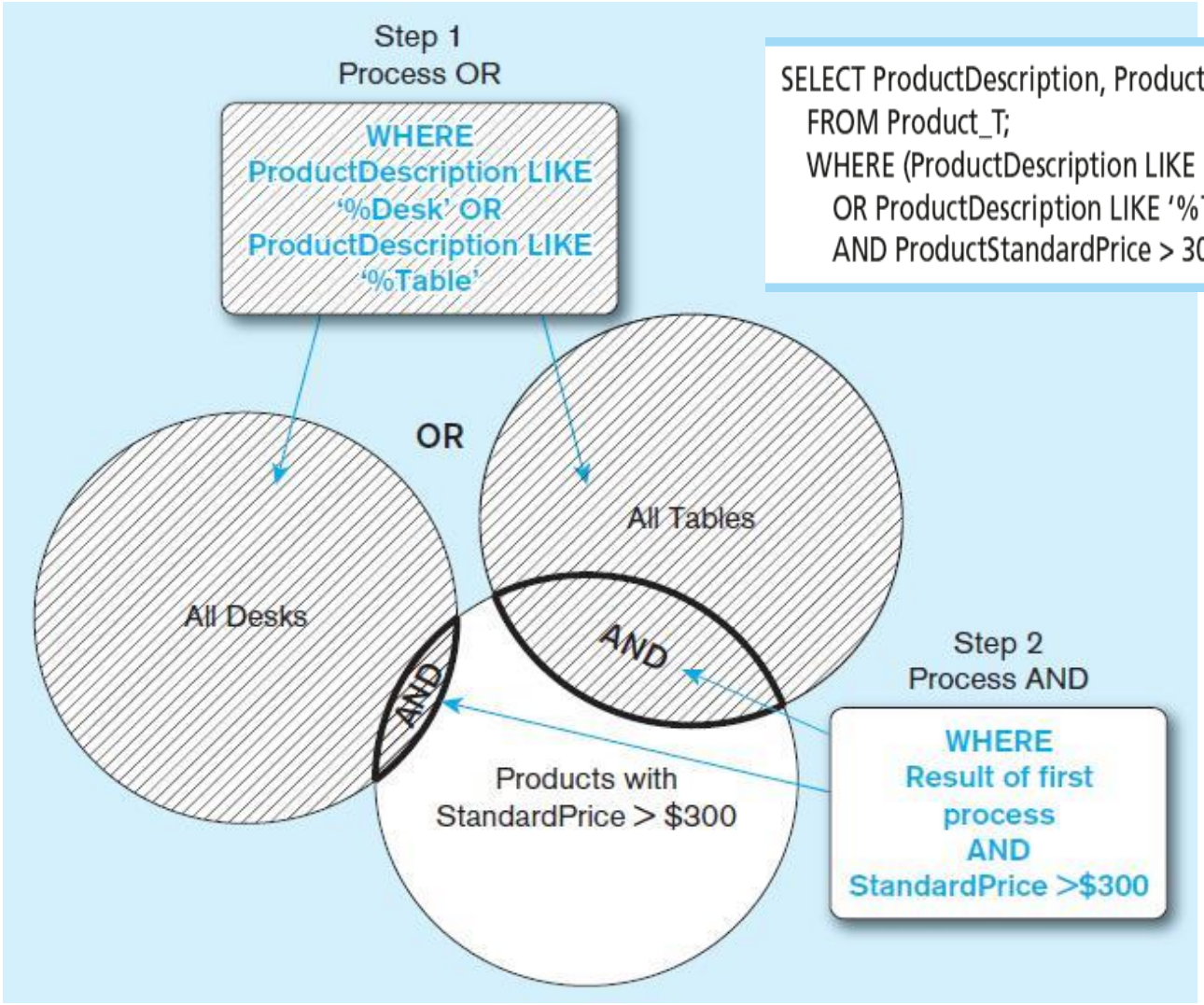
SELECT Example–Boolean Operators

- **With parentheses…these override the normal precedence of Boolean operators**

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

With parentheses, you can override normal precedence rules. In this case parentheses make the OR take place before the AND.

Figure 6-8 Boolean query B with use of parentheses




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

Sorting Results with ORDER BY Clause

- Sort the results first by STATE, and within a state by the CUSTOMER NAME

```
SELECT CustomerName, CustomerCity, CustomerState
   FROM Customer_T
      WHERE CustomerState IN ('FL', 'TX', 'CA', 'HI')
         ORDER BY CustomerState, CustomerName;
```

Note: the IN operator in this example allows you to include rows whose CustomerState value is either FL, TX, CA, or HI. It is more efficient than separate OR conditions.

Categorizing Results Using GROUP BY Clause

- For use with aggregate functions
  - ***Scalar aggregate***: single value returned from SQL query with aggregate function
  - ***Vector aggregate***: multiple values returned from SQL query with aggregate function (via GROUP BY)

```
SELECT CustomerState, COUNT (CustomerState)
   FROM Customer_T
      GROUP BY CustomerState;
```

Qualifying Results by Categories Using the HAVING Clause

✖For use with GROUP BY

```
SELECT CustomerState, COUNT (CustomerState)
   FROM Customer_T
      GROUP BY CustomerState
      HAVING COUNT (CustomerState) > 1;
```

Like a WHERE clause, but it operates on groups (categories), not on individual rows. Here, only those groups with total numbers greater than 1 will be included in final result.

Using and Defining Views

- Views provide users controlled access to tables
- Base Table–table containing the raw data
- Virtual Table–constructed automatically as needed; not maintained as real data
- Dynamic View
  - A “virtual table” created dynamically upon request by a user
  - No data actually stored; instead data from base table made available to user
  - Based on SQL SELECT statement on base tables or other views
  - Contents materialized as a result of a query

Sample CREATE VIEW

Query: What are the data elements necessary to create an invoice for a customer?
Save this query as a view named Invoice_V.

```
CREATE VIEW Invoice_V AS
   SELECT Customer_T.CustomerID, CustomerAddress, Order_T.OrderID,
   Product_T.ProductID,ProductStandardPrice,
   OrderedQuantity, and other columns as required
      FROM Customer_T, Order_T, OrderLine_T, Product_T
         WHERE Customer_T.CustomerID = Order_T.CustomerID
            AND Order_T.OrderID = OrderLine_T.OrderD
            AND Product_T.ProductID = OrderLine_T.ProductID;
```

Advantages of Views

- Simplify query commands
- Assist with data security (but don't rely on views for security, there are more important security measures)
- Enhance programming productivity
- Contain most current base table data
- Use little storage space
- Provide customized view for user
- Establish physical data independence

Disadvantages of Views

- Use processing time each time view is referenced
- May or may not be directly updateable

Create

- An SQL relation is defined using the **create table** command:

  create table r (A_1 D_1, A_2 D_2, ..., A_n D_n,
  (integrity-constraint$_1$),
  ...,
  (integrity-constraint$_k$))

  - r is the name of the relation
  - each A_i is an attribute name in the schema of relation r
  - D_i is the data type of values in the domain of attribute A_i

- Example:

  create table *branch*
  (*branch_name* char(15) **not null,**
  branch_city char(30),
  assets integer)

Integrity Constraints in Create Table

- **not null**
- **primary key** (A_1, ..., A_n)

Example: Declare *branch_name* as the primary key for *branch*

.

create table *branch*
(*branch_name* char(15)**,**
branch_city char(30),
assets integer,
primary key (*branch_name*))

Drop and Alter Table Constructs

- The **drop table** command deletes all information about the dropped relation from the database.
- The **alter table** command is used to add attributes to an existing relation:

  alter table *r* **add** *A D*

  where *A* is the name of the attribute to be added to relation *r* and *D* is the domain of *A.*
  - All tuples in the relation are assigned *null* as the value for the new attribute.
- The **alter table** command can also be used to drop attributes of a relation:

  alter table *r* **drop** *A*

  where *A* is the name of an attribute of relation *r*
  - Dropping of attributes not supported by many databases

Basic Query Structure

- SQL is based on set and relational operations with certain modifications and enhancements
- A typical SQL query has the form:

select $A_1, A_2, \ldots, A_n$
from $r_1, r_2, \ldots, r_m$
where P

  - A_i represents an attribute
  - R_i represents a relation
  - P is a predicate.
- This query is equivalent to the relational algebra expression.

$$\prod_{A_1, A_2, \ldots, A_n} (\sigma_P (r_1 \times r_2 \times \ldots \times r_m))$$

- The result of an SQL query is a relation.

The select Clause

- The **select** clause list the attributes desired in the result of a query
  - corresponds to the projection operation of the relational algebra
- Example: find the names of all branches in the *loan* relation:

  select *branch_name*
  from *loan*
- In the relational algebra, the query would be:

  $\Pi_{branch\_name}(loan)$
- NOTE: SQL names are case insensitive (i.e., you may use upper- or lower-case letters.)
  - E.g. *Branch_Name* ≡ *BRANCH_NAME* ≡ *branch_name*
  - Some people use upper case wherever we use bold font.

Cont....

- SQL allows duplicates in relations as well as in query results.
- To force the elimination of duplicates, insert the keyword **distinct** after select**.**
- Find the names of all branches in the *loan* relations, and remove duplicates

  select distinct *branch_name*
  from *loan*

- The keyword **all** specifies that duplicates not be removed.

  select all *branch_name*
  from *loan*

The select Clause (Cont.)

- An asterisk in the select clause denotes “all attributes”

  select *
  from *loan*

- The **select** clause can contain arithmetic expressions involving the operation, +, –, ∗, and /, and operating on constants or attributes of tuples.
- The query:

  select *loan_number, branch_name, amount* ∗ 100
  from *loan*

  would return a relation that is the same as the *loan* relation, except that the value of the attribute *amount* is multiplied by 100.

The where Clause

- The **where** clause specifies conditions that the result must satisfy
  - Corresponds to the selection predicate of the relational algebra.
- To find all loan number for loans made at the Perryridge branch with loan amounts greater than $1200.

  select *loan_number*
  from *loan*
  where *branch_name* = 'Perryridge' **and** *amount* > 1200
- Comparison results can be combined using the logical connectives **and, or,** and **not.**
- Comparisons can be applied to results of arithmetic expressions.

The where Clause (Cont.)

- SQL includes a **between** comparison operator
- Example: Find the loan number of those loans with loan amounts between \$90,000 and \$100,000 (that is, $\geq$ \$90,000 and $\leq$ \$100,000)

```
select loan_number
        from loan
        where amount between 90000 and 100000
```

The from Clause

- The **from** clause lists the relations involved in the query
  - Corresponds to the Cartesian product operation of the relational algebra.
- Find the Cartesian product *borrower X loan*

```
select *
from borrower, loan
```

- Find the name, loan number and loan amount of all customers having a loan at the Perryridge branch.

```
select customer_name, borrower.loan_number, amount
        from borrower, loan
        where   borrower.loan_number = loan.loan_number  and
                branch_name = 'Perryridge'
```

The Rename Operation

- The SQL allows renaming relations and attributes using the **as** clause:

  old-name **as** *new-name*

- Find the name, loan number and loan amount of all customers; rename the column name *loan_number* as *loan_id.*

select *customer_name, borrower.loan_number* **as** *loan_id, amount*
from *borrower, loan*
where *borrower.loan_number = loan.loan_number*

String Operations

- SQL includes a string-matching operator for comparisons on character strings. The operator "like" uses patterns that are described using two special characters:
  - percent (%). The % character matches any substring.
  - underscore (_). The _ character matches any character.
- Find the names of all customers whose street includes the substring "Main".

```
select customer_name
from customer
where customer_street like '% Main%'
```

- Match the name "Main%"

```
like 'Main\%' escape '\'
```

Ordering the Display of Tuples

- List in alphabetic order the names of all customers having a loan in Perryridge branch

  select distinct *customer_name*
  from *borrower, loan*
  where *borrower loan_number = loan.loan_number* **and**
  branch_name = 'Perryridge'
  order by *customer_name*

- We may specify **desc** for descending order or **asc** for ascending order, for each attribute; ascending order is the default.
  - Example: **order by** *customer_name* **desc**

Set Operations

- Find all customers who have a loan, an account, or both:

  (select *customer_name* **from** *depositor*)
  union
  (select *customer_name* **from** *borrower)*

- Find all customers who have both a loan and an account.

  (**select** *customer_name* **from** *depositor*)
  intersect
  (select *customer_name* **from** *borrower)*

- Find all customers who have an account but no loan.

  (**select** *customer_name* **from** *depositor*)
  except
  (select *customer_name* **from** *borrower)*

Aggregate Functions

- These functions operate on the set of values of a column of a relation, and return a value

avg: average value
min: minimum value
max: maximum value
sum: sum of values
count: number of values

Aggregate Functions (Cont.)

- Find the average account balance at the Perryridge branch.

select avg *(balance)*
from *account*
where *branch_name* = 'Perryridge'

- Find the number of tuples in the *customer* relation.

select count (*) **from** *customer*

- Find the number of depositors in the bank.

select count (distinct *customer_name)* **from** *depositor*

Aggregate Functions – Group By

- Find the number of depositors for each branch.

select *branch_name,* **count (distinct** *customer_name)*
from *depositor, account*
where *depositor.account_number = account.account_number*
group by *branch_name*

Note: Attributes in **select** clause outside of aggregate functions must appear in **group by** list

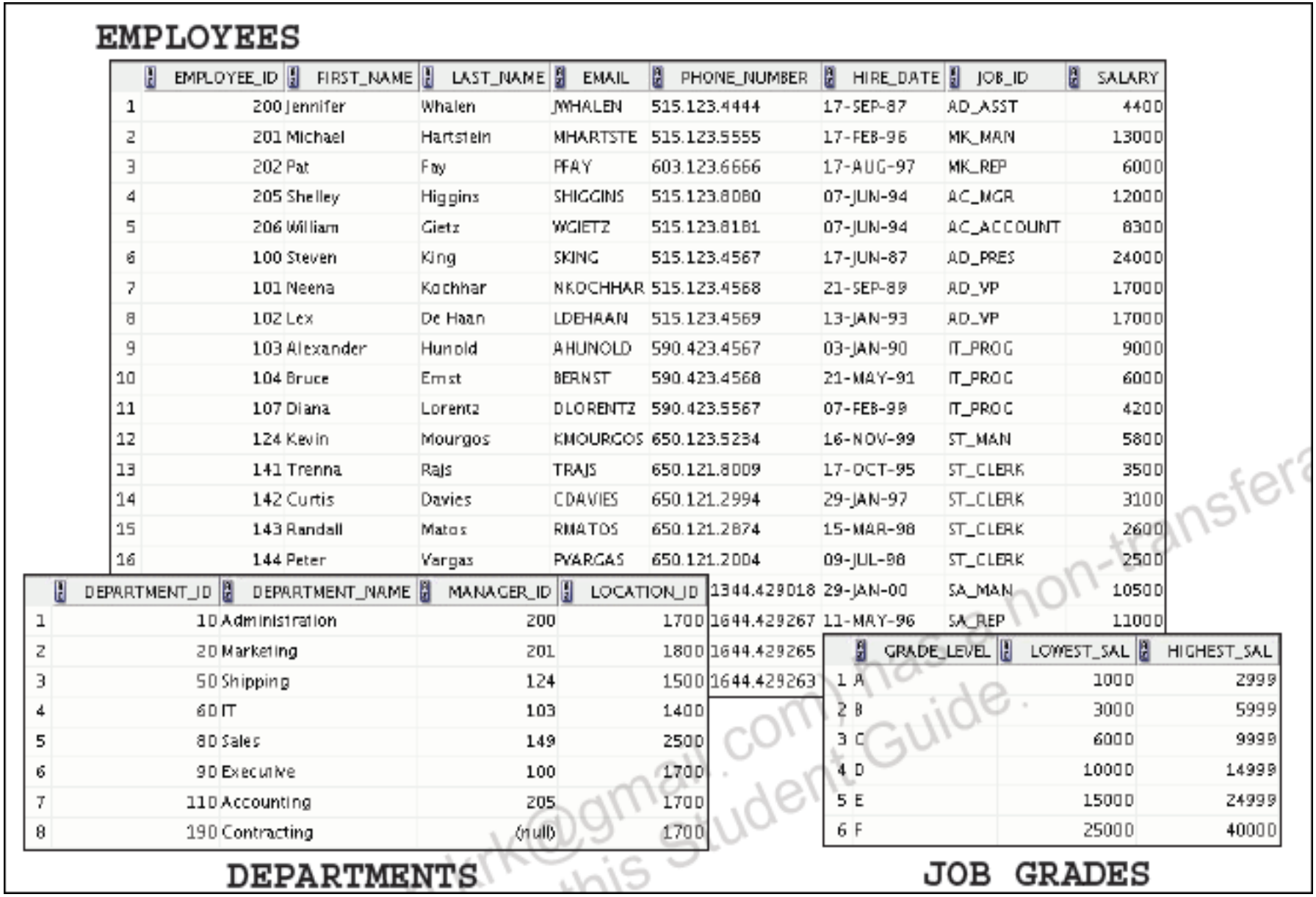
Aggregate Functions – Having Clause

- Find the names of all branches where the average account balance is more than $1,200.

select *branch_name,* **avg** (*balance*)
from *account*
group by *branch_name*
having avg (*balance*) > 1200

Note: predicates in the **having** clause are applied after the formation of groups whereas predicates in the **where** clause are applied before forming groups

Example



EMPLOYEES

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
1	200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	4400
2	201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-96	MK_MAN	13000
3	202	Pat	Fay	PFAY	603.123.6666	17-AUG-97	MK_REP	6000
4	205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	12000
5	206	William	Gietz	WGIETZ	515.123.8181	07-JUN-94	AC_ACCOUNT	8300
6	100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000
7	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000
8	102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000
9	103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000
10	104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	6000
11	107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-99	IT_PROG	4200
12	124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-99	ST_MAN	5800
13	141	Trenna	Rajs	TRAJS	650.121.8009	17-OCT-95	ST_CLERK	3500
14	142	Curtis	Davies	CDAVIES	650.121.2994	29-JAN-97	ST_CLERK	3100
15	143	Randall	Matos	RMATOS	650.121.2874	15-MAR-98	ST_CLERK	2600
16	144	Peter	Vargas	PVARGAS	650.121.2004	09-JUL-98	ST_CLERK	2500
					1344.429018	29-JAN-00	SA_MAN	10500
					1644.429267	11-MAY-96	SA_REP	11000
					1644.429265			
					1644.429263			

DEPARTMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	50	Shipping	124	1500
4	60	IT	103	1400
5	80	Sales	149	2500
6	90	Executive	100	1700
7	110	Accounting	205	1700
8	190	Contracting	(null)	1700

JOB GRADES

	GRADE_LEVEL	LOWEST_SAL	HIGHEST_SAL
1	A	1000	2999
2	B	3000	5999
3	C	6000	9999
4	D	10000	14999
5	E	15000	24999
6	F	25000	40000

```
SELECT last_name, hire_date, salary
FROM   employees;
```

	LAST_NAME	HIRE_DATE	SALARY
1	Whalen	17-SEP-87	4400
2	Hartstein	17-FEB-96	13000
3	Fay	17-AUG-97	6000
4	Higgins	07-JUN-94	12000
5	Gietz	07-JUN-94	8300
6	King	17-JUN-87	24000
7	Kochhar	21-SEP-89	17000
8	De Haan	13-JAN-93	17000

```
SELECT location_id, department_id
FROM   departments;
```

	LOCATION_ID	DEPARTMENT_ID
1	1700	10
2	1800	20
3	1500	50
4	1400	60

Arithematic expression

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide

```
SELECT last_name, salary, salary + 300
FROM   employees;
```

	LAST_NAME	SALARY	SALARY+300
1	Whalen	4400	4700
2	Hartstein	13000	13300
3	Fay	6000	6300
4	Higgins	12000	12300
5	Gietz	8300	8600
6	King	24000	24300
7	Kochhar	17000	17300
8	De Haan	17000	17300
9	Hunold	9000	9300
10	Ernst	6000	6300

Arithematic expression...

```
SELECT last_name, salary, 12*salary+100
FROM   employees;
```

1

	LAST_NAME	SALARY	12*SALARY+100
1	Whalen	4400	52900
2	Hartstein	13000	156100
3	Fay	6000	72100

...

```
SELECT last_name, salary, 12*(salary+100)
FROM   employees;
```

2

	LAST_NAME	SALARY	12*(SALARY+100)
1	Whalen	4400	54000
2	Hartstein	13000	157200
3	Fay	6000	73200

...

concatenate

```
SELECT   last_name||job_id AS "Employees"
FROM     employees;
```

	Employees
1	AbelSA_REP
2	DaviesST_CLERK
3	De HaanAD_VP
4	ErnstIT_PROG

Duplicate Row

The default display of queries is all rows, including duplicate rows.

1

```
SELECT department_id
FROM   employees;
```

	DEPARTMENT_ID
1	10
2	20
3	20
4	110
5	110

2

```
SELECT DISTINCT department_id
FROM   employees;
```

	DEPARTMENT_ID
1	(null)
2	20
3	90
4	110
5	50

where

Using the WHERE Clause

```
SELECT employee_id, last_name, job_id, department_id
FROM   employees
WHERE  department_id = 90 ;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	100	King	AD_PRES	90
2	101	Kochhar	AD_VP	90
3	102	De Haan	AD_VP	90

```
SELECT last_name, job_id, department_id
FROM   employees
WHERE  last_name = 'Whalen' ;
```

	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	Whalen	AD_ASST	10

Comparison Condition

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to
BETWEEN ...AND...	Between two values (inclusive)
IN(set)	Match any of a list of values
LIKE	Match a character pattern
IS NULL	Is a null value

Between

```
SELECT last_name, salary
FROM   employees
WHERE  salary BETWEEN 2500 AND 3500 ;
```

Lower limit Upper limit

	LAST_NAME	SALARY
1	Rajs	3500
2	Davies	3100
3	Matos	2600
4	Vargas	2500

IN

```
SELECT employee_id, last_name, salary, manager_id
FROM   employees
WHERE  manager_id IN (100, 101, 201) ;
```

	EMPLOYEE_ID	LAST_NAME	SALARY	MANAGER_ID
1	201	Hartstein	13000	100
2	101	Kochhar	17000	100
3	102	De Haan	17000	100
4	124	Mourgos	5800	100
5	149	Zlotkey	10500	100
6	200	Whalen	4400	101
7	205	Higgins	12000	101
8	202	Fay	6000	201

Like

- Use the `LIKE` condition to perform wildcard searches of valid search string values.
- Search conditions can contain either literal characters or numbers:
  - `%` denotes zero or many characters.
  - `_` denotes one character.

```
SELECT   first_name
FROM     employees
WHERE    first_name LIKE 'S%' ;
```

	FIRST_NAME
1	Shelley
2	Steven

Like – Wildcard Char

```
SELECT employee_id, last_name, job_id
FROM   employees WHERE  job_id LIKE '%SA\_%' ESCAPE '\';
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID
1	149	Zlotkey	SA_MAN
2	174	Abel	SA_REP
3	176	Taylor	SA_REP
4	178	Grant	SA_REP

Null condition

- Test for nulls with the IS NULL operator

```
SELECT last_name, manager_id
FROM   employees
WHERE  manager_id IS NULL ;
```

	LAST_NAME	MANAGER_ID
1	King	(null)

```
SELECT last_name, job_id, commission_pct
FROM   employees
WHERE  commission_pct IS NULL;
```

	LAST_NAME	JOB_ID	COMMISSION_PCT
1	Whalen	AD_ASST	(null)
2	Hartstein	MK_MAN	(null)
3	Fay	MK_REP	(null)
4	Higgins	AC_MGR	(null)
...			
16	Vargas	ST_CLERK	(null)

Logical Condition

Operator	Meaning
AND	Returns TRUE if *both* component conditions are true
OR	Returns TRUE if *either* component condition is true
NOT	Returns TRUE if the following condition is false

Logical Condition- AND / OR

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary >=10000
AND    job_id LIKE '%MAN%' ;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	201	Hartstein	MK_MAN	13000
2	149	Zlotkey	SA_MAN	10500

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary >= 10000
OR     job_id LIKE '%MAN%' ;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	201	Hartstein	MK_MAN	13000
2	205	Higgins	AC_MGR	12000
3	100	King	AD_PRES	24000
4	101	Kochhar	AD_VP	17000
5	102	De Haan	AD_VP	17000
6	124	Mourgos	ST_MAN	5800
7	149	Zlotkey	SA_MAN	10500
8	174	Abel	SA_REP	11000

Logical Condition- NOT

```
SELECT last_name, job_id
FROM   employees
WHERE  job_id
       NOT IN ('IT_PROG', 'ST_CLERK', 'SA_REP') ;
```

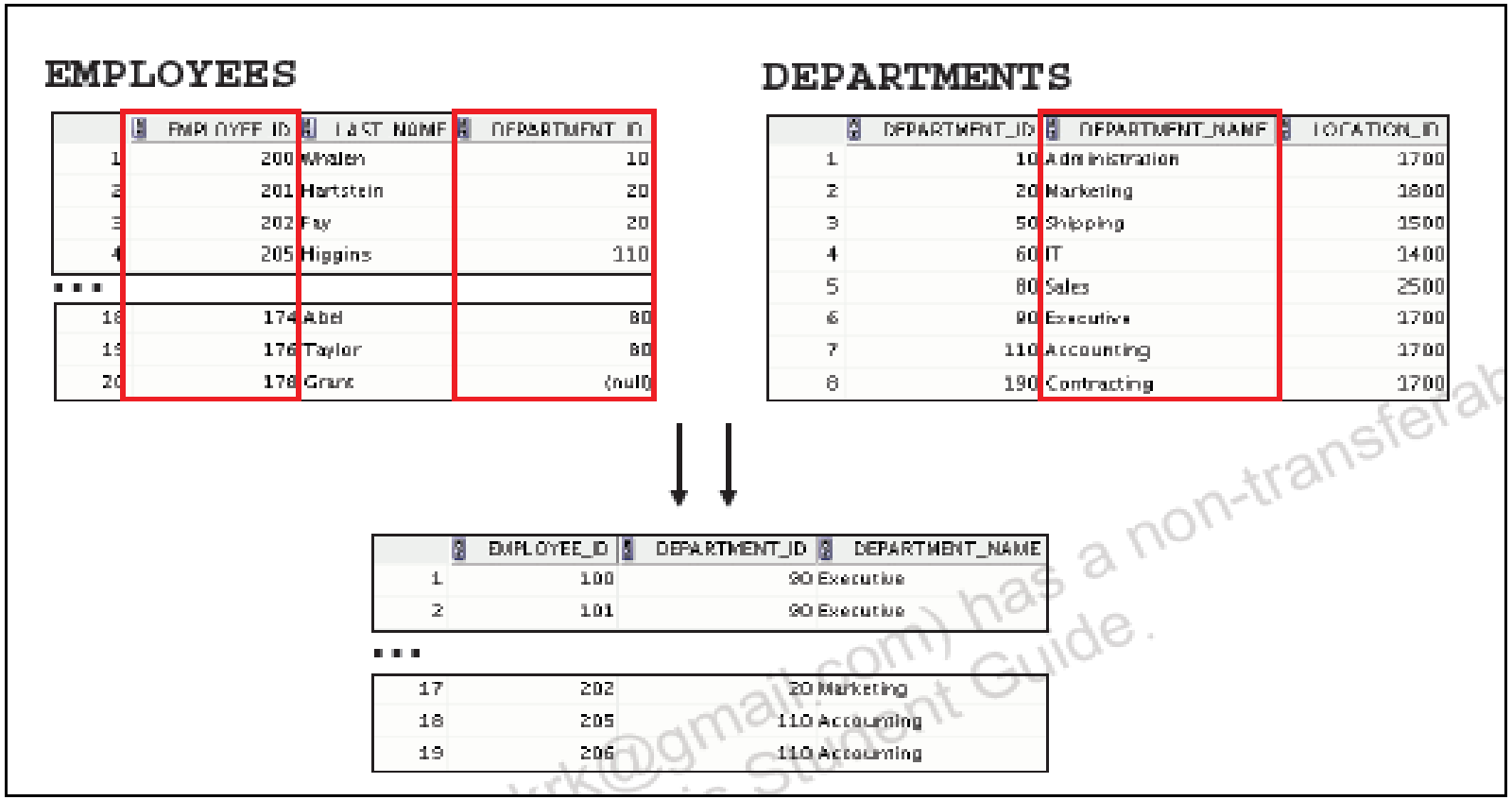
	LAST_NAME	JOB_ID
1	De Haan	AD_VP
2	Fay	MK_REP
3	Gietz	AC_ACCOUNT
4	Hartstein	MK_MAN
5	Higgins	AC_MGR
6	King	AD_PRES
7	Kochhar	AD_VP
8	Mourgos	ST_MAN
9	Whalen	AD_ASST
10	Zlotkey	SA_MAN

Order By

- Sort retrieved rows with the ORDER BY clause
  - ASC: ascending order, default
  - DESC: descending order

```
SELECT   last_name, job_id, department_id, hire_date
FROM     employees
ORDER BY hire_date ;
```

	LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
1	King	AD_PRES	90	17-JUN-87
2	Whalen	AD_ASST	10	17-SEP-87
3	Kochhar	AD_VP	90	21-SEP-89
4	Hunold	IT_PROG	60	03-JAN-90
...				
20	Zlotkey	SA_MAN	80	29-JAN-00


EMPLOYEES
EMPLOYEE_ID
LAST_NAME
DEPARTMENT_ID
200 Whalen 10
201 Hartstein 20
202 Fay 20
205 Higgins 110
174 Abel 80
176 Taylor 80
178 Grant (null)
DEPARTMENTS
DEPARTMENT_ID
DEPARTMENT_NAME
LOCATION_ID
10 Administration 1700
20 Marketing 1800
50 Shipping 1500
60 IT 1400
80 Sales 2500
90 Executive 1700
110 Accounting 1700
190 Contracting 1700
EMPLOYEE_ID
DEPARTMENT_ID
DEPARTMENT_NAME
100 90 Executive
101 90 Executive
202 20 Marketing
205 110 Accounting
206 110 Accounting

Join

- Cross joins
- Natural joins
- USING clause
- Full (or two-sided) outer joins
- Arbitrary join conditions for outer joins

```
SELECT department_id, department_name,
       location_id, city
FROM   departments
NATURAL JOIN locations ;
```

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	CITY
1	60	IT	1400	Southlake
2	50	Shipping	1500	South San Francisco
3	10	Administration	1700	Seattle
4	90	Executive	1700	Seattle
5	110	Accounting	1700	Seattle
6	190	Contracting	1700	Seattle
7	20	Marketing	1800	Toronto
8	80	Sales	2500	Oxford

Constraints

- Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency.

  - NOT NULL
  - UNIQUE
  - PRIMARY KEY
  - FOREIGN KEY
  - CHECK

Constraint...

Data Integrity Constraints

Constraint	Description
NOT NULL	Specifies that the column cannot contain a null value
UNIQUE	Specifies a column or combination of columns whose values must be unique for all rows in the table
PRIMARY KEY	Uniquely identifies each row of the table
FOREIGN KEY	Establishes and enforces a foreign key relationship between the column and a column of the referenced table
CHECK	Specifies a condition that must be true

Not Null Constraint

- Declare *branch_name* for *branch* is **not null**

  branch_name **char**(15) **not null**

Ensures that null values are not permitted for the column:

	EMPLOYEE_ID	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID
1	176	Grant	KGRANT	011.44.1644.429263	24-MAY-99	SA_REP	7000	(null)
2	206	Gietz	WGIETZ	515.123.8181	07-JUN-94	AC_ACCOUNT	8300	110
3	205	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	12000	110
4	100	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000	90
5	102	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000	90

...

NOT NULL constraint (No row can contain a null value for this column.)

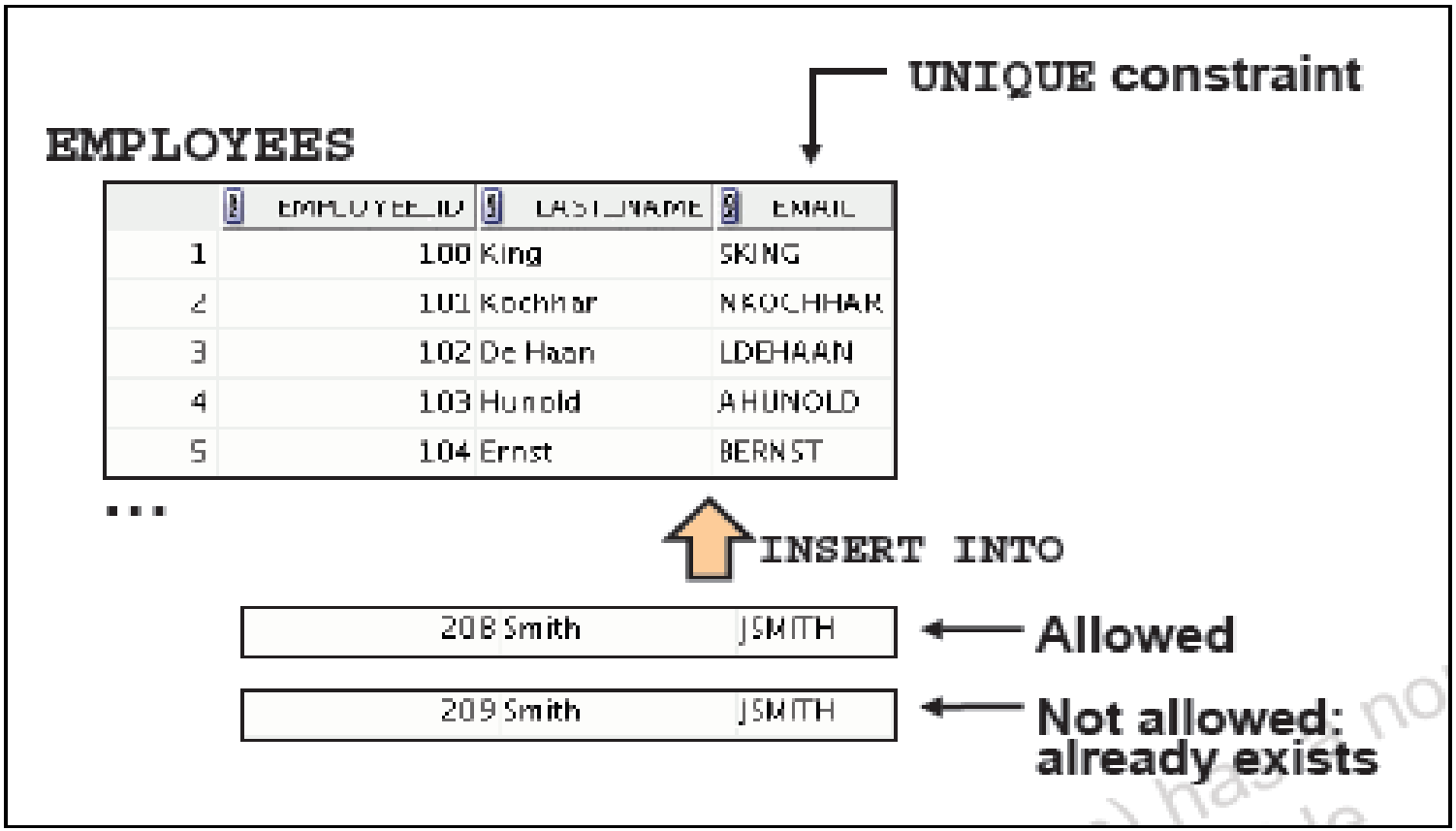
NOT NULL constraint

Absence of NOT NULL constraint (Any row can contain a null value for this column.)

The Unique Constraint

- **unique** (A_1, A_2, …, A_m)
- The unique specification states that the attributes

  $A1$, $A2$, … Am form a candidate key.
- Candidate keys are permitted to be null (in contrast to primary keys).



The check clause

- **check** (*P*), where *P* is a predicate

  Example: Declare *branch_name* as the primary key for *branch* and ensure that the values of *assets* are non-negative.

```
create table branch
        (branch_name    char(15),
         branch_city    char(30),
         assets         integer,
         primary key (branch_name),
         check (assets >= 0))
```

The check clause (Cont.)

- The **check** clause in SQL-92 permits domains to be restricted:
  - Use **check** clause to ensure that an hourly_wage domain allows only values greater than a specified value.

    create domain *hourly_wage* **numeric(5,2)**
    constraint *value_test* **check**(*value* > = 4.00)
  - The domain has a constraint that ensures that the hourly_wage is greater than 4.00
  - The clause **constraint** *value_test* is optional; useful to indicate which constraint an update violated.

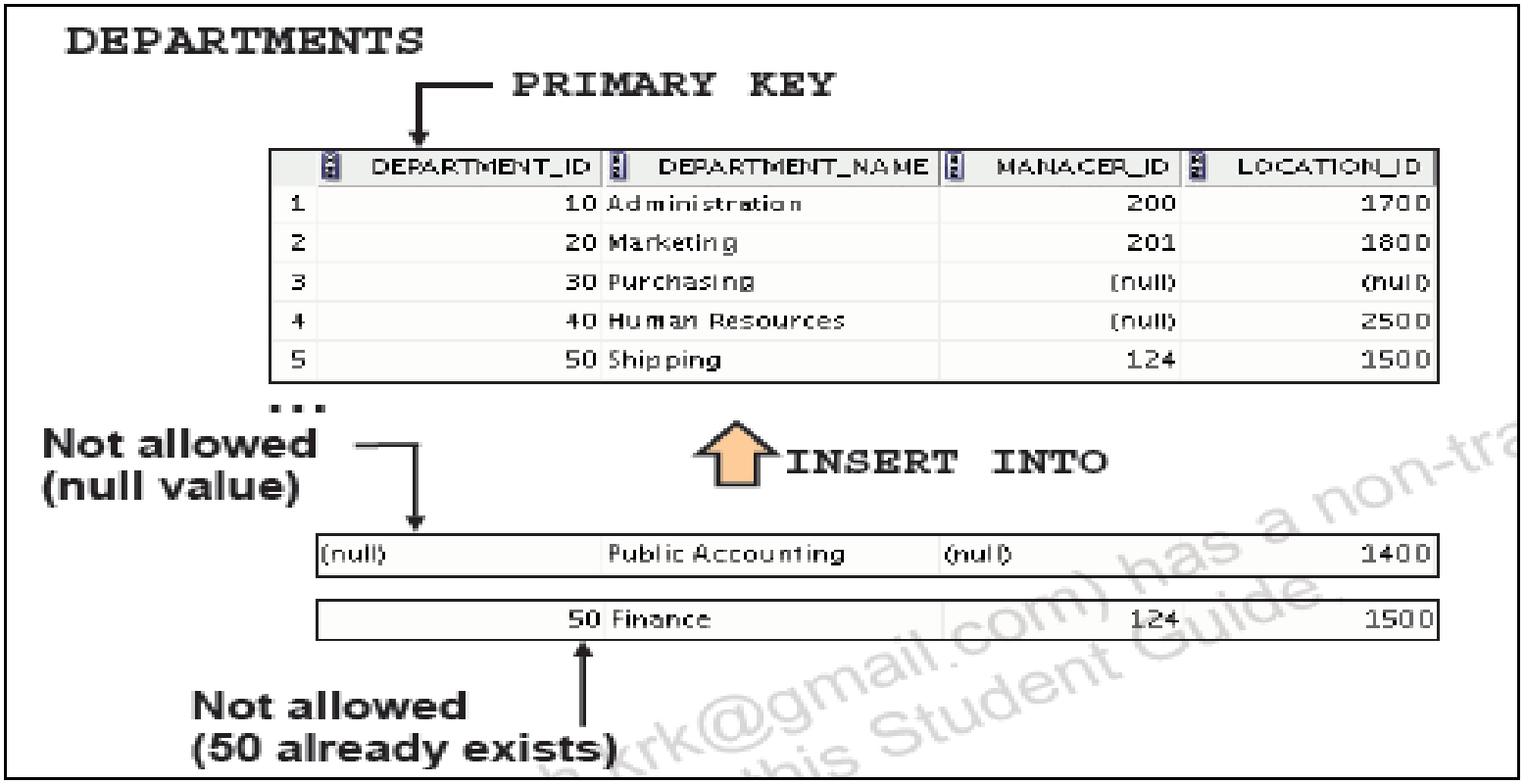
Referential Integrity

- Ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation.
- Primary and candidate keys and foreign keys can be specified as part of the SQL **create table** statement:
  - The primary key clause lists attributes that comprise the primary key.
  - The unique key clause lists attributes that comprise a candidate key.
  - The foreign key clause lists the attributes that comprise the foreign key and the name of the relation referenced by the foreign key. By default, a foreign key references the primary key attributes of the referenced table.

Referential Integrity-Primary key

create table *customer* (*customer_name* **char**(20)**,** *customer_street* **char**(30), *customer_city* **char**(30), **primary key** (*customer_name*))

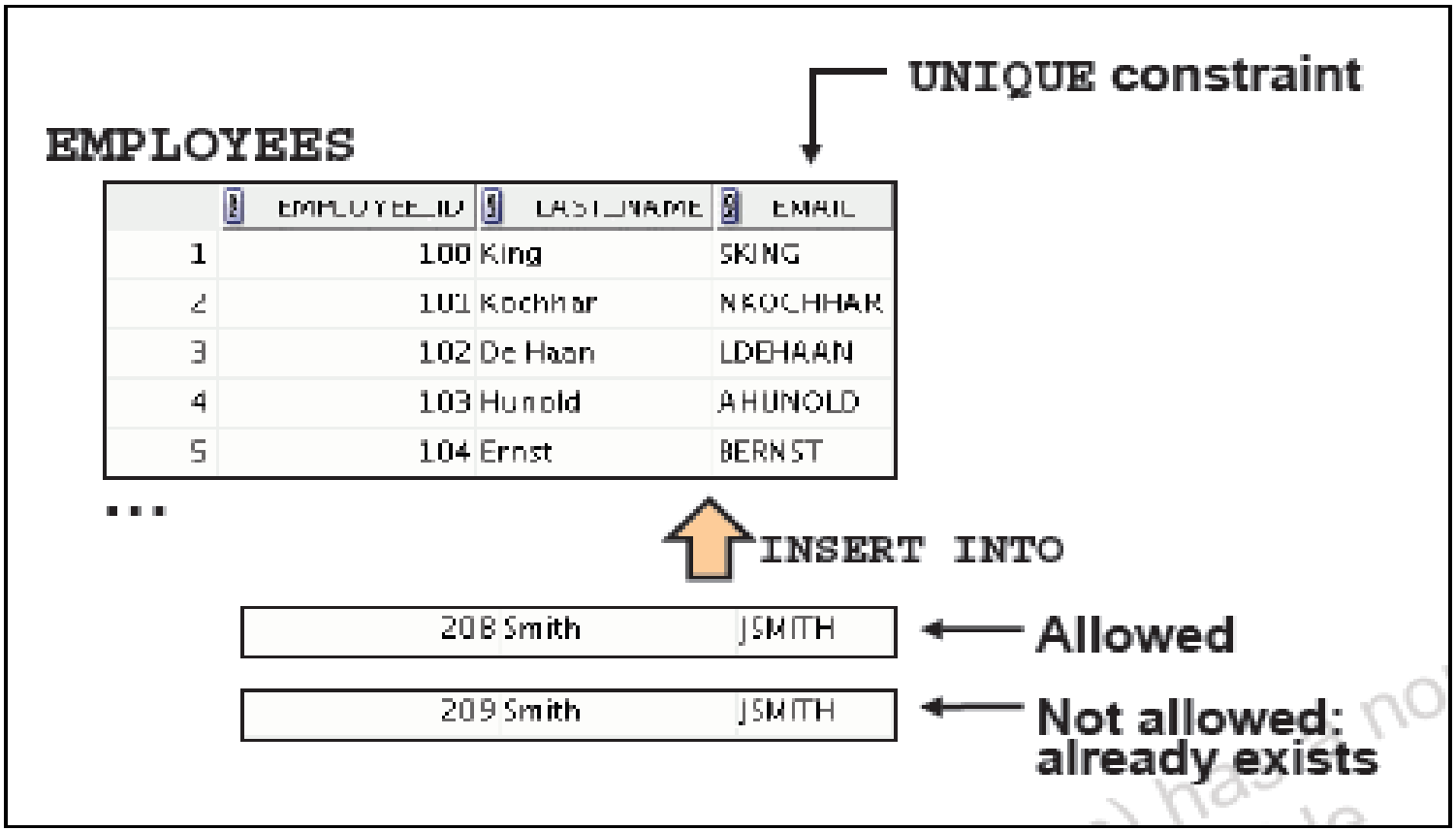
create table *branch* (branch_name **char**(15)**,** *branch_city* **char**(30), *assets* **numeric**(12,2), **primary key** (*branch_name*))



The Unique Constraint

- **unique** (A_1, A_2, …, A_m)
- The unique specification states that the attributes

  $A1$, $A2$, … Am form a candidate key.
- Candidate keys are permitted to be null (in contrast to primary keys).



Referential Integrity in SQL – Foreign key

create table *account*
(*account_number* **char**(10)**,**
branch_name **char**(15),
balance **integer**,
primary key (*account_number),*
foreign key (*branch_name*) **references** *branch*)

create table *depositor*
(*customer_name* **char**(20)**,**
account_number **char**(10)**,**
primary key (*customer_name, account_number),*
foreign key (*account_number*) **references** *account,*
foreign key (*customer_name*) **references** *customer*)

Foreign key

- FOREIGN KEY: Defines the column in the child table at the table-constraint level
- REFERENCES: Identifies the table and column in the parent table
- ON DELETE CASCADE: Deletes the dependent rows in the child table when a row in the parent table is deleted
- ON DELETE SET NULL: Converts dependent foreign key values to null

Truncate

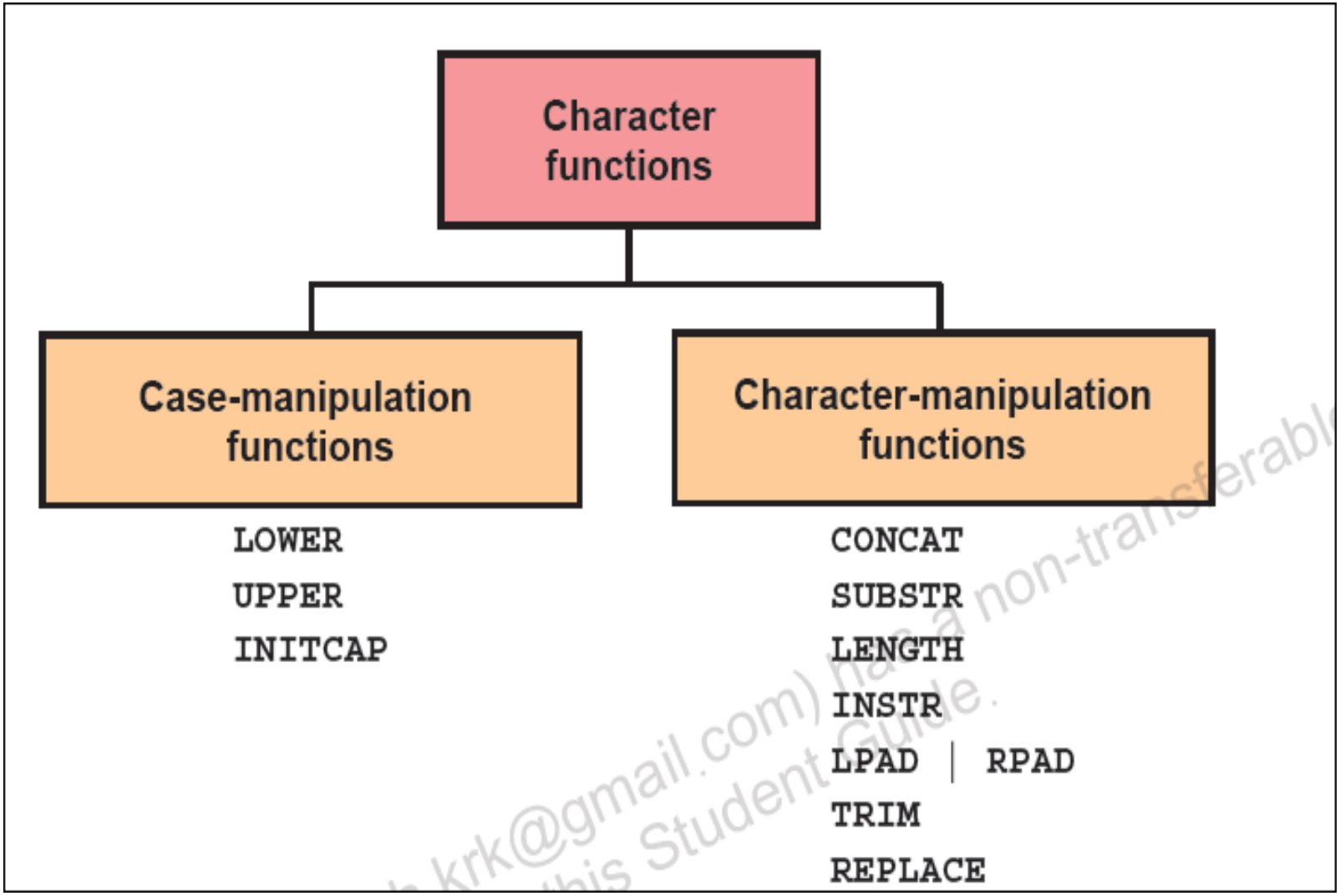
- Removes all rows from a table, leaving the table empty and the table structure intact
- Is a data definition language (DDL) statement rather than a DML statement; cannot easily be undone
- Syntax:

```
TRUNCATE TABLE table_name;
```

- Example:

```
TRUNCATE TABLE copy_emp;
```

String function



String function....

Function	Purpose
LOWER(*column|expression*)	Converts alpha character values to lowercase
UPPER(*column|expression*)	Converts alpha character values to uppercase
INITCAP(*column|expression*)	Converts alpha character values to uppercase for the first letter of each word; all other letters in lowercase
CONCAT(*column1|expression1, column2|expression2*)	Concatenates the first character value to the second character value; equivalent to concatenation operator (||)
SUBSTR(*column|expression,m[,n]*)	Returns specified characters from character value starting at character position *m*, *n* characters long (If *m* is negative, the count starts from the end of the character value. If *n* is omitted, all characters to the end of the string are returned.)

String function....

Function	Purpose
LENGTH(*column|expression*)	Returns the number of characters in the expression
INSTR(*column|expression, 'string', [,m], [n]*)	Returns the numeric position of a named string. Optionally, you can provide a position *m* to start searching, and the occurrence *n* of the string. *m* and *n* default to 1, meaning start the search at the beginning of the search and report the first occurrence.
LPAD(*column|expression, n, 'string'*) RPAD(*column|expression, n, 'string'*)	Pads the character value right-justified to a total width of *n* character positions Pads the character value left-justified to a total width of *n* character positions
TRIM(*leading|trailing|both, trim_character FROM trim_source*)	Enables you to trim heading or trailing characters (or both) from a character string. If *trim_character* or *trim_source* is a character literal, you must enclose it in single quotation marks. This is a feature that is available in Oracle8*i* and later versions.
REPLACE(*text, search_string, replacement_string*)	Searches a text expression for a character string and, if found, replaces it with a specified replacement string

String function- case manupulation function

Function	Result
LOWER('SQL Course')	sql course
UPPER('SQL Course')	SQL COURSE
INITCAP('SQL Course')	Sql Course

```
SELECT 'The job id for '||UPPER(last_name)||' is '
 ||LOWER(job_id) AS "EMPLOYEE DETAILS"
FROM   employees;
```

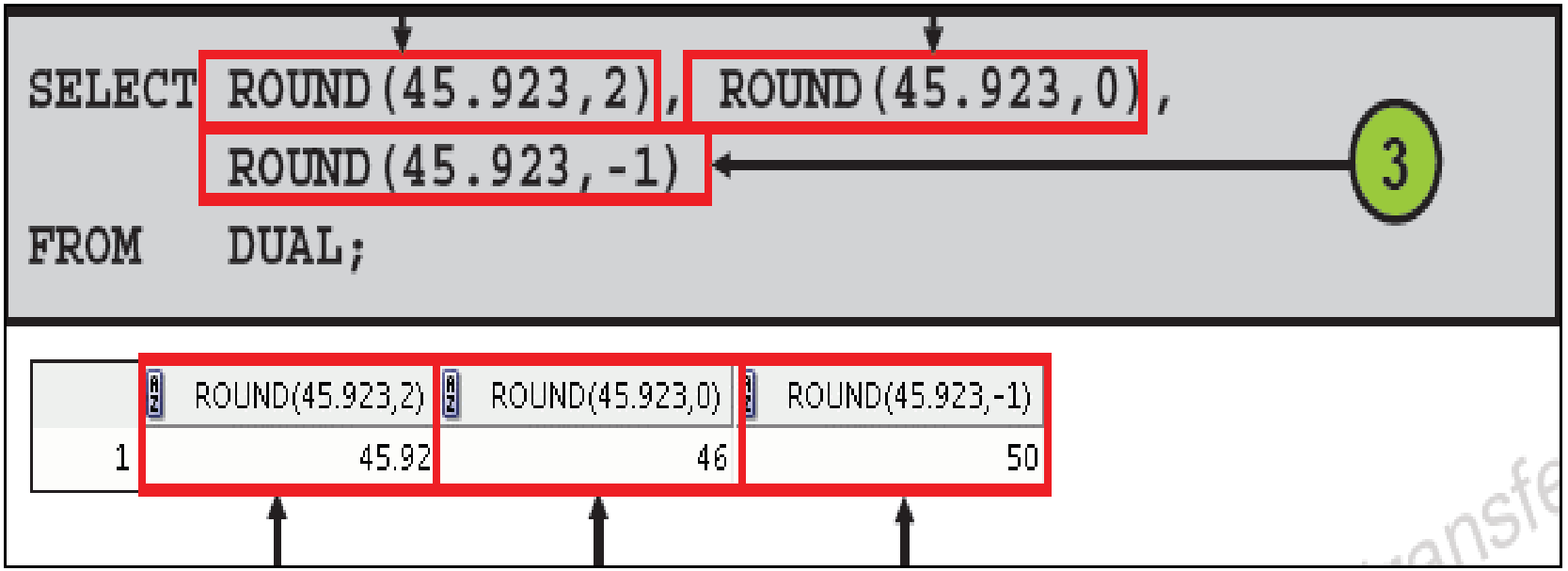
	EMPLOYEE DETAILS
1	The job id for ABEL is sa_rep
2	The job id for DAVIES is st_clerk
3	The job id for DE HAAN is ad_vp

String function- case manipulation function

Function	Result
CONCAT('Hello', 'World')	HelloWorld
SUBSTR('HelloWorld',1,5)	Hello
LENGTH('HelloWorld')	10
INSTR('HelloWorld', 'W')	6
LPAD(salary,10,'*')	*****24000
RPAD(salary, 10, '*')	24000*****
REPLACE ('JACK and JUE','J','BL')	BLACK and BLUE
TRIM('H' FROM 'HelloWorld')	elloWorld

Number function

Function	Result
ROUND(45.926, 2)	45.93
TRUNC(45.926, 2)	45.92
MOD(1600, 300)	100



Date function

- SYSDATE is a function that return date.

Display the current date using the DUAL table.

```
SELECT  SYSDATE
FROM    DUAL;
```

	SYSDATE
1	06-NOV-08

```
SELECT last_name, hire_date
FROM   employees
WHERE  hire_date < '01-FEB-88';
```

	LAST_NAME	HIRE_DATE
1	Whalen	17-SEP-87
2	King	17-JUN-87

Date function

Operation	Result	Description
date + number	Date	Adds a number of days to a date
date – number	Date	Subtracts a number of days from a date
date – date	Number of days	Subtracts one date from another
date + number/24	Date	Adds a number of hours to a date

Date function....

```
SELECT last_name, (SYSDATE-hire_date)/7 AS WEEKS
FROM   employees
WHERE  department_id = 90;
```

	LAST_NAME	WEEKS
1	King	1116.148574735449735449735449735449735545
2	Kochhar	998.00571759259259259259259259259259593
3	De Haan	825.148574735449735449735449735449735545

Function	Result
MONTHS_BETWEEN ('01-SEP-95','11-JAN-94')	19.6774194
ADD_MONTHS ('11-JAN-94',6)	'11-JUL-94'
NEXT_DAY ('01-SEP-95','FRIDAY')	'08-SEP-95'
LAST_DAY ('01-FEB-95')	'28-FEB-95'

Date function....

Assume SYSDATE = '25-JUL-03':

Function	Result
ROUND(SYSDATE,'MONTH')	01-AUG-03
ROUND(SYSDATE ,'YEAR')	01-JAN-04
TRUNC(SYSDATE ,'MONTH')	01-JUL-03
TRUNC(SYSDATE ,'YEAR')	01-JAN-03

TO_CHAR function

```
SELECT last_name,
       TO_CHAR(hire_date, 'fmDD Month YYYY')
       AS HIREDATE
FROM   employees;
```

	LAST_NAME	HIREDATE
1	Whalen	17 September 1987
2	Hartstein	17 February 1996
3	Fay	17 August 1997
4	Higgins	7 June 1994
5	Gietz	7 June 1994
...		
19	Taylor	24 March 1998
20	Grant	24 May 1999

Subquery/ Nested query

```
SELECT   select_list
FROM     table
WHERE    expr operator
                    (SELECT      select_list
                    FROM         table);
```

Subquery....

```
SELECT last_name, salary
FROM   employees
WHERE  salary >
                (SELECT salary
                 FROM   employees
                 WHERE  last_name = 'Abel');
```

11000

	LAST_NAME	SALARY
1	Hartstein	13000
2	Higgins	12000
3	King	24000
4	Kochhar	17000
5	De Haan	17000

Subquery....

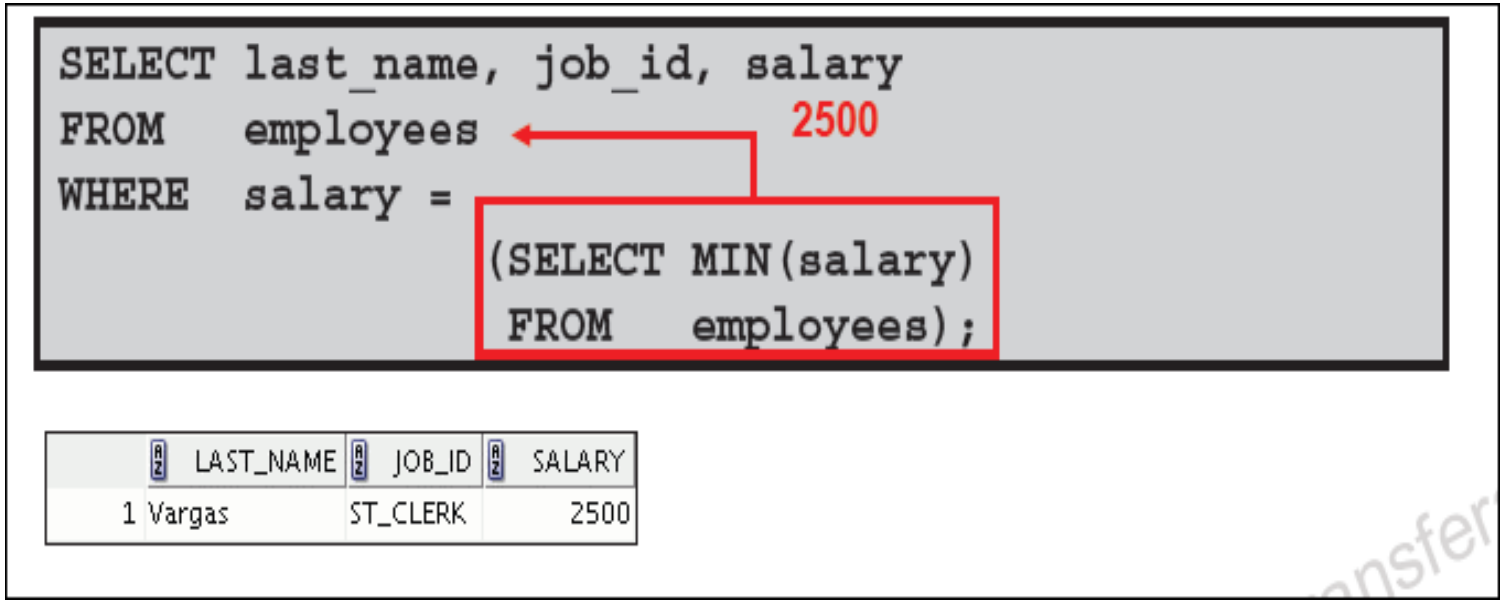
Display the employees whose job ID is the same as that of employee 141:

```
SELECT last_name, job_id
FROM   employees
WHERE  job_id =
                (SELECT job_id
                 FROM   employees
                 WHERE  employee_id = 141);
```

	LAST_NAME	JOB_ID
1	Rajs	ST_CLERK
2	Davies	ST_CLERK
3	Matos	ST_CLERK
4	Vargas	ST_CLERK

Subquery....

Display last name ,job_id ,salary in which salary is greater than min salary.



Subquery....

Operator	Meaning
IN	Equal to any member in the list
ANY	Compare value to each value returned by the subquery
ALL	Compare value to every value returned by the subquery

```
SELECT last_name, salary, department_id
FROM   employees
WHERE  salary IN (SELECT   MIN(salary)
                  FROM     employees
                  GROUP BY department_id);
```

Subquery

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ANY
                (SELECT salary
                 FROM   employees
                 WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

9000, 6000, 4200

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	144	Vargas	ST_CLERK	2500
2	143	Matos	ST_CLERK	2600
...				
9	206	Gietz	AC_ACCOUNT	8300
10	176	Taylor	SA_REP	8600

Subquery- All

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ALL
                (SELECT salary
                 FROM   employees
                 WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

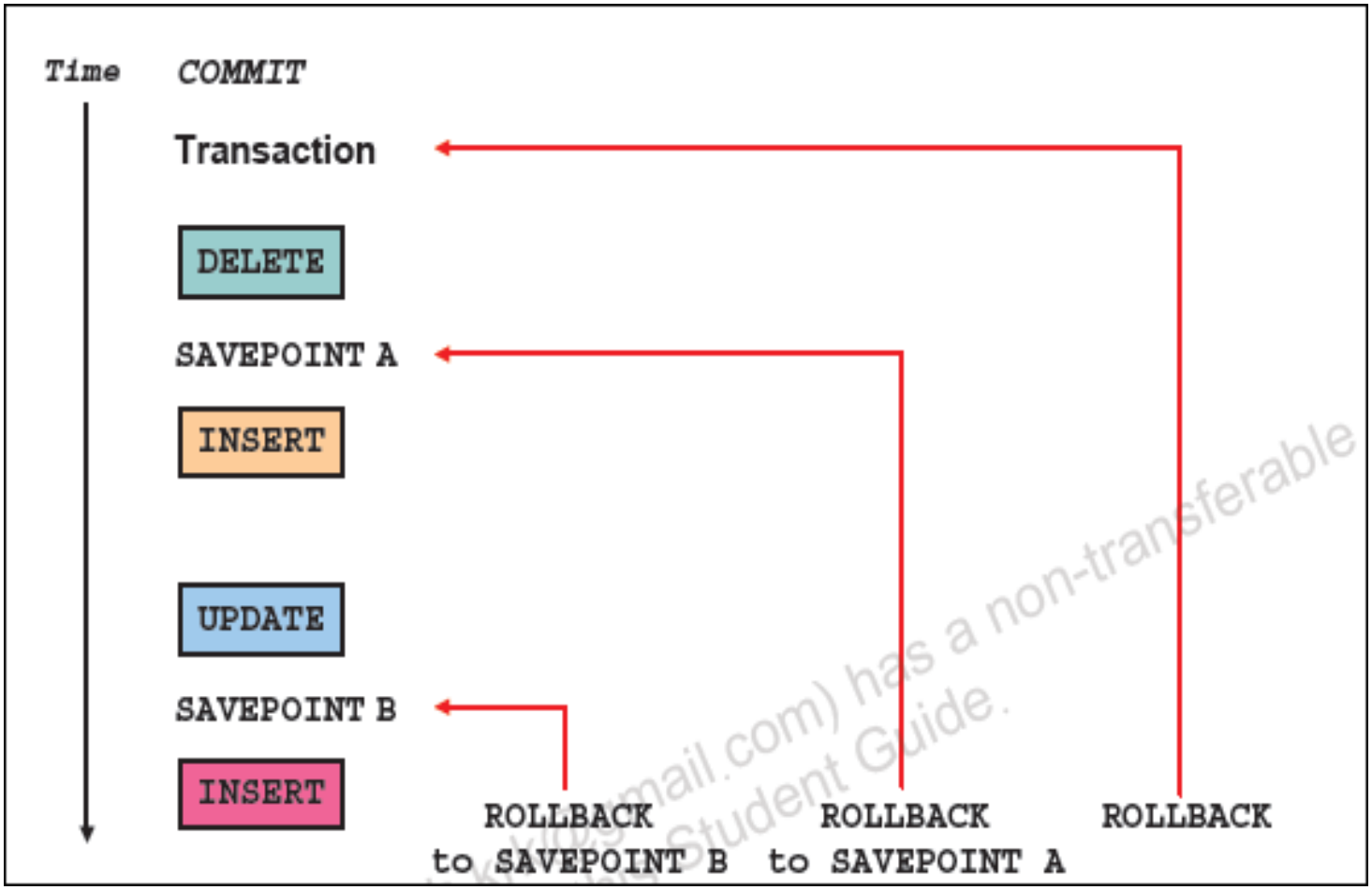
9000, 6000, 4200

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	141	Rajs	ST_CLERK	3500
2	142	Davies	ST_CLERK	3100
3	143	Matos	ST_CLERK	2600
4	144	Vargas	ST_CLERK	2500

Transaction Control Command

- Commit
- Rollback
- Savepoint

Transaction Control Command



Transaction Control Command...

Statement	Description
COMMIT	Ends the current transaction by making all pending data changes permanent
SAVEPOINT *name*	Marks a savepoint within the current transaction
ROLLBACK	ROLLBACK ends the current transaction by discarding all pending data changes.
ROLLBACK TO *SAVEPOINT name*	ROLLBACK TO SAVEPOINT rolls back the current transaction to the specified savepoint, thereby discarding any changes and or savepoints that were created after the savepoint to which you are rolling back. If you omit the TO SAVEPOINT clause, the ROLLBACK statement rolls back the entire transaction. Because savepoints are logical, there is no way to list the savepoints that you have created.

Transaction Control Command-COMMIT

- Make the changes:

```
DELETE FROM employees
WHERE  employee_id = 99999;
1 rows deleted

INSERT INTO departments
VALUES (290, 'Corporate Tax', NULL, 1700);
1 rows inserted
```

- Commit the changes:

```
COMMIT;
Commit complete
```

ROLLBACK

Discard all pending changes by using the ROLLBACK statement:

- Data changes are undone.
- Previous state of the data is restored.
- Locks on the affected rows are released.

```
DELETE FROM copy_emp;
20 rows deleted
ROLLBACK ;
Rollback complete
```

SAVEPOINT

- It is used to roll back to the savepoint marker.

COMMIT	Makes all pending changes permanent
SAVEPOINT	Is used to roll back to the savepoint marker
ROLLBACK	Discards all pending data changes

Data Control Language

- Grant
- Revoke

It gives or removes access rights to the structures within it

Grant

- The grant statement is used to confer authorization

  grant <privilege list>

  on <relation name or view name> **to** <user list>
- <user list> is:
  - a user-id
  - **public**, which allows all valid users the privilege granted
  - A role
- Granting a privilege on a view does not imply granting any privileges on the underlying relations.
- The grantor of the privilege must already hold the privilege on the specified item (or be the database administrator).

Revoke

- The **revoke** statement is used to revoke authorization.

  revoke <privilege list>

  on <relation name or view name> **from** <user list>

- Example:

  revoke select on *branch* **from** U_1, U_2, U_3

- <privilege-list> may be **all** to revoke all privileges the revokee may hold.
- If <revokee-list> includes **public,** all users lose the privilege except those granted it explicitly.
- If the same privilege was granted twice to the same user by different grantees, the user may retain the privilege after the revocation.
- All privileges that depend on the privilege being revoked are also revoked.

Queries

1. **Consider following schema and represent given statements in relation algebra form.**

   *** Branch(branch_name,branch_city)**

   *** Account(branch_name, acc_no, balance)**

   ***Depositor(Customer_name, acc_no)**

   (i) Find out list of customer who have account at ‘abc’ branch.

   (ii) Find out all customer who have account in ‘Ahmedabad’ city and balance is greater than 10,000.

   (iii) Find out list of all branch name with their maximum balance.

Queries

- **Implement following relation using SQL query.**

  Student(stud_no,stud_name,sub1,sub2,totalmark, percentage). Create the table, add 5 records and display the data.

- **Update the mark of sub1 of student_no=111 with 50 and also Calculate totalmark and percentage accordingly.**

Queries

- Implement following relation using SQL query.

  Employee(emp_no, emp_name, department, city, salary)

  (1) Find all the employee whose emp_no is lessthan 100 and salary more than 25000 and department is “Account”

  (2) count the no of employee and Sum the salary of all employee

  (3) Delete the employee having minimum salary.

queries

- Solve following queries with following table, where underlined attribute is primary key.
- Person(ss#, name, address)
- Car(license, year, model)
- Accident(date, driver, damage-amount)
- Owns(ss#, license)
- Log(license, date, driver)

1. Find the name of a person whose license number is '12345'.

3. Add a new accident by 'Ravi' for 'BMW' car on 01/01/2013 for damage amount of 1.5 lakh rupees.